An Informed Search for the WWW using a Hybrid Implementation of A*

By Daniel J. Sullivan

For I460/I461 Senior Thesis Requirements

ABSTRACT: This work explores the application of the A* heuristic search function to the problem of document retrieval and classification based upon a relevancy criterion. This work includes a modification of A* and proposes a means of determining relevancy as a function of independent textual mappings.

Table of Contents

[1] A* is pronounced “a star” for purposes of discussion.

1.0 – Background

The topic of this work builds upon the history of intelligent systems in the context of internet and explores these related areas as a prelude to the main problem – the search for useful and relevant information. In providing background for this story, this work will limit the discussion of this topic to those relevant events taking place between the end of WWII and the present time. A thorough exploration of the history of Information Retrieval (IR) would begin with Aristotle and catalog the wider history of indexing and classification - which would be beyond the scope of this work.[2] For this work, however, just enough of the history will be presented.

1.1 – AI and Expert Systems

Allen Turing, considered to be one of the principal founding fathers of computer science, began thinking about computers outside the context of computation. Before Turing and during his professional career, automated systems for information processing were in their infancy. Systems like the ENIAC (the first electronic computer) and Charles Babbage's calculating machine (developed a century earlier) were mainly directed at solving arithmetic problems. ENIAC[3] was built to calculate trajectories for artillery, which was a human-resource intensive process up to that point.[4] Babbage was fundamentally interested in the same problem – the Royal Navy needed to calculate navigation tables and Babbage saw this process as something a machine could accomplish. Turing provided computer science with the theoretical foundation for progress, and his idealization of a computer (the turing machine) is in many ways the first system for searching information.[5] Turing saw potential far beyond mere calculation, and so in many ways he is also the father of artificial intelligence, or AI[6].

Artificial Intelligence would not be used as a term until 1956, when John McCarthy first used the term in a letter submitted to his peers – a letter which proposed a conference on the matter of AI. AI began as an exploration of human-like problem solving and learning –

[2] [Belew, 2000, pg. xv]
[3] ENIAC (electronic numerical integrator and computer), was designed and constructed during WWII. It was only programmable in the sense that you could rewire its logic circuits to solve new problems.
[4] [Luger, 2002, pgs. 10-11]
[5] Turing machines are a mathematical abstraction, but in most ways they represent the basic and critical components of any modern computer (this is especially interesting, since this model was developed in the 1930's). Since they operate with an external storage (or tape) they also have some of the aspects of the search agent about them. The universe of a turing machine is simplified, but it is a powerful metaphor for any information processing system.
[6] Turing proposed the now famous test (or thought experiment) which would help a researcher determine whether a computer is intelligent. A person at a terminal, after a process of asking questions to two unknown agents (one computer and one human), is asked to decide which agent was human and which was not. If the person does not know the difference or makes an incorrect choice (according to Turing) you must grant that the computer is intelligent.

languages such as LISP[7] were developed to assist in the process. By the late 1960's a subset of AI research was beginning to grow – the first **expert systems**[8] were proposed, designed and programmed.

An expert system is a program that internalizes and uses some domain specific **knowledge base**.[9] It can be programmed in almost any language, but languages have been designed almost solely for the development of expert systems.[10] Expert systems are limited to a narrow problem domain. A knowledge engineer (expert system designer) can gather as much explicit and implicit knowledge as possible, from domain experts (e.g. Doctors, Mechanics, and Military Professionals), and program the computer with strategies for finding the correct solution to a particular problem. The computer is restricted to whatever knowledge base it possesses and with the exception of probabilistic reasoning it cannot infer anything that is not in its knowledge base. Expert systems can appear to be creative problem solvers, but they are really procedural problem solvers and cannot make inferences outside of their narrowly defined domain.[11]

1.2 – Intelligent Agents and their Users

During the 1980's, changes in AI, computer science and other fields began affecting our understanding of human problem solving, some of this revolved around the field of **cognitive science**.[12] Fuzzy Logic, for instance, proposed a system of approximate reasoning where solutions were not solely true or false. Chaos Theory described a world of complex (and apparently random) interactions which enabled at least the perception of stability. And, greater understanding of the architecture of the human brain gave scientists an alternative to the previously popular symbol/algorithm focus of problem solving. Neural Networks would impact this quest in two ways: first by changing the focus of algorithm development and second by showing that part of the problem may be the lack of parallelism in the architecture itself. A few

[7] LISP (List Processing) is a language which is purely functional in nature. To achieve iteration, a function must "call-itself" (recursion).
[8] An expert system is a computer program designed to make correct decisions within a narrow problem domain, and it is expected that its decisions will be as good as a human agent. Many of these systems are in fact collections of subsystems (databases, programs, OS) that are designed to work together to produce intelligent answers to problems. Often these systems are programmed in specialized knowledge representation languages like PROLOG.
[9] You can think of a knowledge base as a set of rules and facts. These rules and facts enable a computer to reason logically about a problem and to derive conclusions that are similar to those which humans would derive. Within the small world of the computer's knowledge base, the computer can be quite competent at a number of different tasks.
[10] PROLOG (Programming Logic) was developed during the late 1970's and early 1980's. A version of this system runs on UNIX platforms today called XSB. The syntax of this language is
[11] [Luger, 2002, pgs. 258-262]
[12] Cognitive science is an extension of what began as cognitive psychology. This trend began almost 50 years ago during the period when behaviorism (stimulus – response) had its greatest influence. Instead of taking the black-box approach to human intelligence, cognitive researchers are interested in the internal representations and processes by which intelligent minds solve problems.

researchers began to think more in terms of **artificial life**.[13] This view was further affected by the coming of pervasive networks, most notably the arrival of the WWW[14] (World Wide Web). The web presented, and still presents, an almost ideal environment within which to place **intelligent agents**.[15]

Intelligent agents are software designed to assist users and also act on their behalf. They are like expert systems in that they are narrow in scope and targeted to a particular user/problem domain; however, they are designed to learn from the user and they are designed to be semi-autonomous.[16] Where the expert systems of the past were thought of as completely autonomous, these systems are seen in the context of their relation to human beings and how they collaborate with these human users. To formally state the distinction; expert systems mimic human thought and do so statically with explicit decision rules within a specific domain. Intelligent agents learn from people and do so dynamically using various design techniques. They are envisioned as partners in problem solving.[17]

1.3 – Web Spiders[18]

A web spider is a special kind of agent. Actually, it is designed to be much more autonomous than some other kinds of agents. Search engine providers, such as Google[19], design spiders to find new links and retrieve enough information to enable some kind of indexing by the search engine. The nature of this work is in some ways reminiscent of Babbage's calculating machine – it is repetitive, and at times formulaic. A better analogy may be that of the late 19th Century factory worker who simply sits next to a lever and pulls it periodically. Apparently, computers are not yet conscious, and are appropriate for this kind of boring work.

Internet search companies like Yahoo developed solutions to this problem during the web boom of the late 90's. Their business model required them to solicit users to submit the user's

[13] Artificial Life is a general category of experimentation based upon researcher methods which simulate the environment and constraints of living organisms. In these cases, it is evolution which is often the target and the means by which the intelligent systems improve.
[14] The WWW, or World Wide Web, is really a heterogeneous mixture of technologies which enable the sharing of files and access to servers. It began solely as a means of sharing files and linking them together but it has become a world encompassing domain. Since any reasonable description could be a paper by itself, I will avoid going into too much detail here. Important aspects of this history include the history of its progenitors – DARPANET and the internet.
[15] [Maes, 1995, pgs. 1-4]
[16] [Maes, 1994, pgs. 1-7]
[17] [Rhodes and Starner, 1996, pgs. 1-7]
[18] The term web spider is synonymous with web crawler. The reason for calling these systems spiders was because of a notion of these programs traversing the web, on their own, like a spider. This is not what is actually taking place. Web spiders are much more like telephone auto-dialers, with intelligence, and they have a purpose similar to that of the telemarketer or telephone researcher fishing for information about a person. Where telephone researchers want to map the demographic pattern, web spiders seek to map the information domain. Spiders surf the web, but they do so within the narrow confines of what intelligence they possess.
[19] Google is a company specializing in search engine technology.

website address, offer paid web services to companies and individuals (and free web services), and build applications that would discover and help categorize web sites. In addition to this, and other proposed services, these companies constructed (and still construct) complex ontologies in which to place all of this information.

One possible solution to this problem is the use of standardized markup languages which provide additional indications of meaning, like RDF (Resource Description Framework).[20] When the WWW was in its infancy, Tim Berners-Lee (and others) gave us the means to make the technology of the internet more flexible and accessible to ordinary people. There have been many forces which have made access to the WWW more democratic. Because of the yearly decrease in the cost of computing (especially relative to computing power), computers are becoming affordable to even the lowest income Americans. Unlike the printing press, the rich will not monopolize this technology and it is already having the effect of changing the rules of human communication. While all of this expansion of communication and openness is true, rules and standards must be followed to make the system work.[21]

The value of standard markups is beyond the scope of this work, but it is arguable that this standard should be limited to a practical minimum; otherwise we could spend a tremendous amount of time identifying the meanings of these documents and updating these meanings. Ultimately it is akin to a Ptolemaic[22] adjustment and as we consistently extend the labeling, we may be ignoring a more long term solution – the development of effective information agents that assist and cooperate with the user to find relevant information.[23]

Organizations are faced with the same problem which plagues individuals: too much information with few, if any, useful and effective means of prioritizing its use. More standard operating procedures produced by information technology professionals might off-set some of this disorganization. Human beings are creative thinkers. Procedures can be useful in addressing some of these problems, but it is unlikely that the solution to the problem is more structure and restrictions on the creative expression of people.[24]

1.4 – The Problem of Information Discovery

The problem domain of this work is directly related to scholarship. As an example, the work of students researching and writing term papers is tied to the methods and practice of

[20] [WW3C 1999, (internet document)]
[21] [Berners-Lee, 1999]
[22] Ptolemy developed an astronomical theory in ancient times and medieval theologians and philosophers made adjustments to the system so it could keep up with current facts – rather than simply developing a new theory.
[23] [Berners-Lee, 1999, pgs. 177-179] **I think Berners-Lee clearly contends that this is a two pronged strategy, develop standards of markup which enable a **semantic web** and develop effective search agents to traverse and to analyze this environment.
[24] [Fayyad, Piatetsky-Shapiro and Smyth, 1996, pgs. 27-30]

information discovery. Even with the changes the internet has brought, the problem of finding relevant information has not changed that much. Libraries are still necessary and the old-fashioned methods are still important because there is no quality control in cyberspace.[25]

This work addresses some of the design issues involving the abstract idealization of a Search Agent[26] (SA). While certain aspects of this work address specific problems, it is important to keep in mind that the data itself can be treated generically from the standpoint of the SA. The SA is given a sample or a request, and it is the job of the SA to find additional information which is related in some well defined way. Because of the complexity of the subject, and the time constraints of this research project, many interesting aspects of this problem are not explored. Those issues not explored will be left for future work.

2.0 – Statement of the Problem

The general problem domain is finding and retrieving useful information. This work addresses this problem with respect to one kind of WWW medium – HTML documents.[27] This work explores an automated approach to seeking information in the WWW. There are many approaches to this problem, so the scope has been limited to the application of search algorithms developed by AI researchers.

The thesis of this work is the following:

Searching the WWW for relevant information is described in this work as an instance of the graph search problem, and as such it is possible to apply previously developed informed search techniques which use a domain specific heuristic.

In this kind of search there is no clear, single 'goal state'. Without a single identifiable goal state, there may be multiple states or clusters of acceptable states.[28] This entire search is constrained by the amount of information gained and the resources expended. In contrast, this

[25] This is a generalization. Yahoo and Google (and all the others) stake a claim to doing the best job possible. Given the circumstances and the lack of any history of this kind of work it is unfair to be too critical. But the notion of Quality Information is new. To some extent, the Quality revolution has not yet impacted the information revolution.
[26] A search agent is a very general notion, it is specific to the problem of search, but the domain may not be an information space. The domain of search can be any complex set of states, each of which may or may not contain something the agent needs.
[27] HTML is an abbreviation for Hyper Text Markup Language. The development of this formatting language
[28] Acceptable, in this context, can only be defined as a result the user is satisfied with. There are many factors which go into a user's interests and profile which are beyond the scope of this work, but these factors would help to define in a systematic way a reasonable definition of 'acceptable'.

work describes one heuristic search solution which is appropriate to this problem domain and which accommodates the potential of many goal states.

3.0 – Methodology

This section explores the design of a search heuristic used to evaluate best moves for the SA (Search Agent). The set of search algorithms to be investigated are the informed or **heuristic**[29] search techniques. Unlike the DFS (Depth First Search) or BFS (Breadth First Search), informed searches use knowledge about the domain to help in determining the next best move. A* search is one kind of algorithm in this class. A* evaluates next best moves based upon two conditions, the current cost of the search [G] and the expected cost to completion [H] from the current node. However, A* is not intended to work in environments where there is no clearly defined goal state or where there is no fixed or well understood admissibility constraints. Put another way, when human agents search the WWW, they don't always know if they have reached their goal, or they may not immediately understand the value of information found. There may be no way of knowing if a human agent retrieved the most relevant information for the least cost without a great deal of study. To simplify the problem of cost, this work will use time as the primary economic factor in the process of search. Human agents typically use time as a constraint on a search and as a means of evaluating their performance. [30]

3.1 - Goal Nodes for Document Search

[29] Heuristic: basically this is a method or rule of thumb which is not guaranteed to solve the problem, but will solve a class of problems in a domain faster than the algorithmic approach. It is like a shortcut, but often shortcuts can lead us into the wrong direction.

[30] There are many resources being consumed by agents (human or computer) while searching a file space (like the internet). But most of these costs can be accounted for in terms of time. To simplify my implementation of A* I have decided to use time as the principal cost measure accounted for.

The algorithm being discussed uses information gained from an initial sample (or samples) to generate the first set of nodes (WWW based html documents) to visit. What we are interested in are those documents which appear similar to the initial sample; this similarity will be based upon a number of factors. The principal set of factors will be discussed in dept in section 3.3.

What does it means to reach a goal while searching for information? It probably doesn't mean retrieving a single relevant document; otherwise a great mass of information would be discarded.[31] It can be intuitively suggested that there is more than one document published to the internet which would be similar to our sample - with varying degrees of similarity. Clearly, the target of our search is a set of documents which are similar, but is there any reasonable way to determine how large this set is and whether it is all contained in one connected graph? (See Figure 1 below)

Figure 1: One Connected Cluster and Two Disconnected subsets

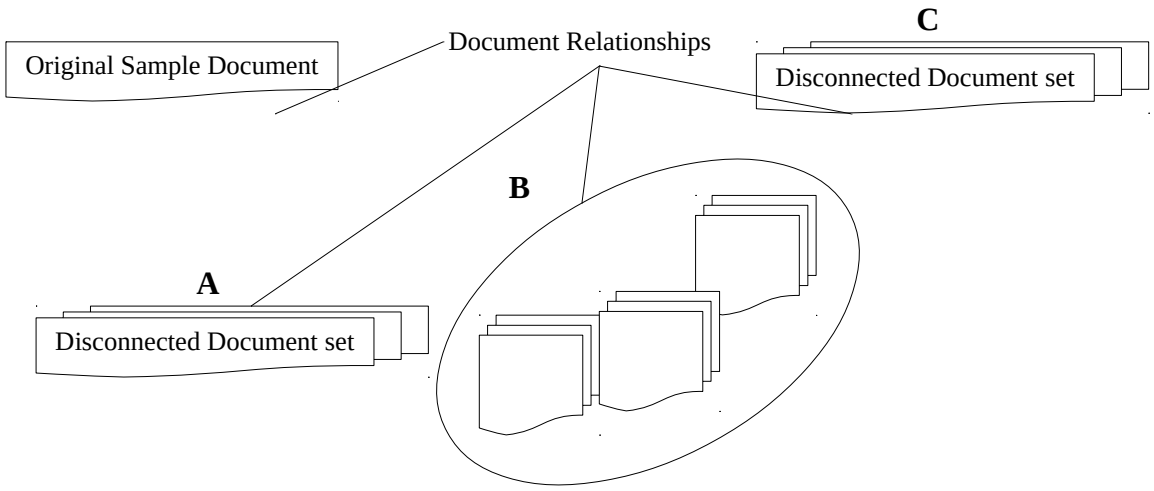


Figure 1 is an abstract description of what may be the state of document linkage in the WWW and so it is conjecture. This work proposes, for this problem domain, that there are large sets (or networks) of files which are connected (associated by some linking mechanism) and contain like information, but the set of all documents similar to a single document is probably not connected (or ordered).[32] This WWW's ordering mechanism is not based on a universal

[31] In this case we are assuming a search where what the agent wants is raw information related to a topic and not necessarily one specific file or document. To find a specific piece of information would be, in many ways, a different problem.

[32] There is a set of all html documents H, and a set of related documents RD $\subset$ H. For RD, there is a set of all links retrieved from elements in RD and this set is built using the link from the document in RD (the link for this page) and the links present on the page as an ordered pair (a,b), we call this set LK. My argument is that there is no relation on LK, where xRx achieves transitive closure. This is a conjecture, but I think it guarantees that there really is no way of simply traversing links and retrieving all related documents. However, since there are multiple

ontology, but rather is a partially ordered heterogeneous set of information. In addition to this, there may not be any guarantee of knowing what set constitutes enough relevant information for the SA to determine it has reached its goal. However, the process of getting this related material seems to be a search in the purest sense – a search without well defined closure. Sets A, B and C in figure 1 may each represent acceptable sets of information or maybe only the union of these would be acceptable; in either case this is not simply a problem of finding a goal state.

The goal state for our proposed SA is the complete retrieval of one set of relevant documents. The following are possible halting conditions for the SA and are not equivalent to goal states where the SA has achieved its best path solution:

1. **Maximum Depth:** this would be calculated as the number of levels of expansion, which in the case of the WWW may not be useful. Considering that the branching factor is both unpredictable and probably unmanageable.

2. **Maximum Number of Visits (downloads):** this would be very straight forward and would allow the user to control the total number of connections. If a visit includes successful and unsuccessful connections then there would be absolute control of the number of steps the SA takes.

3. **Maximum Time Limit:** given the unpredictability of connecting to sites on the internet, this may be the best and most complete method of control. A central time clock can represent total cost and ensure that the SA does not simply get locked up on a particular search.

4. **File Memory Usage Limit:** it may be critical to limit the number of bytes received as a result of a connection. This may not be the best method to control search at the highest level but might indicate a means of control at a lower level to prevent certain processes from overloading file space.

5. **Constant User Supervision and Control:** if the SA is bundled as part of a total computer-user system, then there could be direct control of the agent on a monitoring basis. This kind of control would be related to the others but would involve the user at critical points to select or reject paths, and to potentially halt the SA.

None of the triggers mentioned for halting the program gives a justification related to the problem, they are subjective/context limits based upon the needs of the user. They are simply means of stopping the SA, not of determining why the search should be stopped. The goal of a human agent searching the web is to find as much relevant material as possible, given a finite amount of time. So, the method which provides the most relevant material over time may be the best. The user will not likely know what this value is, but it may not be a constant. Yet, it is

unrelated documents that link these subsets, there is transitive closure on the whole – as we would expect in a bi-partite graph. This is an oversimplification, but if we think of this as a bipartite problem, then to reach transitive closure means to leave the relevant sub-graph and branch into materials less relevant to get back to more relevant material. I say this is a bipartite graph, because we might conceive of two different kinds of nodes, nodes of interest and those that are of no interest. So, there may be a path leading to closure, but it must traverse both kinds of nodes.

conceivable that no system will perform better than perfection. So, maybe we should consider the goal state for an SA not as a place reached but rather as an amount of work performed given the constraint of time.[33]

A perfect system which only locates relevant materials and never gets side-tracked could never do better than the band-width of its perception. Humans, who make mistakes, are limited by the amount of information over time they can process. As part of this heuristic, we are going to assume that the number of bytes over time of all relevant materials will be [(network bandwidth)*time] for a perfect agent. Network bandwidth will vary, so for this example I am going to assume a value of 50 kilobytes per second. This is an arbitrary value, and a better value should be chosen based upon the real capabilities of a network. For this discussion, the following assumptions are made; 1) network bandwidth is 50k/s, 2) a perfect agent would only select related materials, 3) we will work with time in terms of seconds, 4) for a perfect agent, the total information retrieved is roughly bandwidth multiplied by time.

We may therefore assume that the best path for information searching is a value of retrieval which comes very close to the actual bandwidth of the information system. To make this experimental model simpler and more flexible, we will accept real performance that is a significant portion of best-performance and set this as our working best-path. This is the adjustment gamma (Γ) and say that: $\Gamma = 1/x$ where x is in R and is a number greater than or equal to 1. Or, gamma will never be less than 0 or greater than 1. By multiplying gamma by best performance, we end up with a best-performance adjusted that can be modified as a reasonable admissibility constraint.

Best Path Adjusted (BPA) equals (bandwidth * time * Γ), where $0 < \Gamma <= 1$, this should never yield a value better than the perfect performance. This "path" which is being discussed is more like an acceptable performance value, since even if a machine is performing well within our acceptable guidelines, this still does not provide us with a trigger for halting the search and finding a specific "goal".

Consider another assumption, that the user limits the search with the use of a time limit. This time limit may be 10 minutes, it may be 10 hours. In this case, we can use the formula above to calculate the best performance. BPA = (bandwidth * max-time *Γ). Adjust the factor Γ by setting it to ¾ or 75% and set the max time value at 30 minutes. Therefore, for this example, the optimal amount of relevant data would be: BPA = (50000 * 1800 * 0.75) which is BPA = 67.5 megabytes of relevant information. It is important to assume, for the moment, that an SA

[33] [Dennis, Taylor, 2002 pg. 5]

may outperform this value of 67.5 mb based upon values substituted for Γ and therefore we are assuming that the calculation for BPA may not be correct and would not be admissible.[34]

3.2 – A Hybrid A* Implementation of Heuristic Search for Documents

As stated previously, A* works on the basis of adding current and heuristically predicted costs in order to determine which next step will be most optimal. The search agent wants to follow paths which will have the lowest overall cost. The following are components of this calculation:

1. G(N) is a function which gives you the cost of the search up to the node N.

2. H(N) is a evaluation function which gives you an estimated cost to complete the search to the goal state (in our application we are looking for the best next step which will yield the most relevant information). For H(N) to be admissible, it cannot overestimate the cost of reaching the goal state along a maximum path.

3. F(N) = G(N) + H(N), where F(N) provides the SA with the estimated cost of the cheapest path from N (the current node or web-page) to the goal state.

In order to develop an applicable G(N), we need to review some of the domain specific costs involved. What are the costs associated with searching the internet? Some of these could be; total elapsed time at work, processor time used, file space used and total available bandwidth monopolized.

Earlier it was assumed that time is the principal measure of costs for this problem, because all of the resources listed above can be accounted for in terms of time (and later in terms of dollars) we will simplify and assume that time itself is an adequate measure of G(N). This assumption depends on the fact that all of the other costs are a function of time and would be consistently managed resources. So, G(N) is S, where S represents total seconds elapsed during the search.

H(N), for this problem domain, is more difficult to construct.[35] H(N) is obviously a function which relates the current document to a maximum amount of relevant information gained. There is a function for determining how relevant a document is to a search, let's call this R(N,X), where N is the current document (node) being visited and X is the start sample which we are comparing against (X is also the means of beginning the search and can be viewed as a single document of interest). Each document contains links to other related materials; we want to

[34] For the algorithm to be admissible, it should not under-estimate the best case so the actual cost is less than the best cost. For example, if we were to attempt to find the best path between two cities, and we used straight-line distance as our best-case we would not expect to do better. However, what if we found a way around the straight-line distance (this requires an extreme interpretation of what might be physically possible) then the straight-line distance would no longer be admissible.

[35] [Barr and Feigenbaum, 1981, pgs. 67-69]

decide whether or not to include these links in the search. We can see these links as doors and the document as a room (a room containing good or bad information). These doors can be opened or may remain closed, and the value of the current room (document) will determine the value of all of the doors associated with it. We would expect to leave a no-information room and backtrack to other rooms, with better doors, if needed.

If R(N,X) gives us the value of a document (and its associated file links), can we infer that documents linked to N are more likely to lead to the retrieval of the most relevant information for the time constraint? For now, this work assumes this is a true statement.[36] If G(N) = S, where S is the total amount of time elapsed, then (TL – S) will give us the estimated time left, we can call this AT (available time). TL is the time limit set by the user. DS (Document Size) gives the current amount of relevant information retrieved in bytes and DV (Document Value) = R(N,X) * DS.

This example requires some adjustment in our notion of what valuable information is. It may be content, it may be links, it may be a combination of both and therefore a designer of an SA may need to revisit this issue.[37] 67.5 megabytes of relevant material in 30 minutes is the BP (Best Path or Case), if we set gamma = 0.75 and our bandwidth averages to 50k for retrieving information.

If we visit a page which contains 100 kilobytes of information, and has a relevance value of 0.89, then this implies that 89 kilobytes of data retrieved were related to the sample document. CRI (Current Relevant Information) retrieved is 37.5 megabytes for this example. CRI can be a running evaluation of the size of the SA's relevant information cache in bytes.[38]

BP – CRI gives us the amount of work in bytes left to be performed. Or BL (Bytes Left). Which for our example would be 67.5 – 37.5, or 30 megabytes. If we set BL = DV * S, then we can come up with a value in seconds to assign to each link embedded in the web page. This is an estimate of the total number of seconds it would take to reach the best-case defined by BP. If BL = 30,000,000 bytes, and DV = 89,000 bytes, then S = BL/DV. Or, H(N) = (BP – CRI)/DV = S = 337 seconds or 5.6 minutes. If we assume the SA has already used 28 minutes or a total of 30

[36] It is very important to be clear about how big an assumption this is. The WWW contains commercial advertising and embedded links that are often not related to the content of the core material. It is beyond the scope of this work, but it would be interesting to come up with additional filters to remove these links from a 'good' link list based upon further heuristics. Possibly, more complex lexical pattern matching could be used to eliminate obviously non-relevant links, no matter how good the document score was for the page in which they were embedded.

[37] It is beyond the scope of this work to discuss optimal lexical analysis and content analysis. It is within the scope of this work to explore some basic (and primitive) methods for doing this. Relevancy is very complex, and I think it would be interesting to explore the application of ANN (Artificial Neural Networks) to relevancy evaluation in any future work related to this.

[38] This is a file containing all information, or a variable containing a measure in Bytes of relevant information gained. This value helps the SA calculate the necessary amount of work left to be performed given the amount of material retrieved thus far.

minutes allotted for the search, then this node would fail to yield the expected benefit to achieve the goal in best time. But, if these links have a better score than other nodes they will still be selected. F(N) for this node will be, 1800 + 337, or 2137 seconds.

As should be clear, there are a number of assumptions built into my application of the A* heuristic function. Some of these assumptions are reasonable, some are simply expedient (like assuming a link on a page is related to the material on the page). While researching this problem, it has become clear that A* may not be the best strategy. It may be that this is a really good problem domain for Hill Climbing. In its simplest form, hill-climbing says choose the next best path simply based on its heuristic evaluation and not on previous costs. Whether hill-climbing is better is not in the scope of this work, but it is worth noting that it may in fact be a better and more direct strategy.[39]

3.3 – Document Relevance Criteria for Heuristic Search

Determining the relevance of a document can be based upon several factors (or document features). Many of these can be broken down into 3 principal feature classes:

1. Lexical/Statistical: This is simply the statistical analysis of the document. It is more complicated than simply word frequency and there are many sophisticated techniques for doing this. I have adopted the simplest example for purposes of this work.

2. Syntactic: This is an analysis of the grammar or rules of the language being employed in the document. This is a very interesting problem, but also extraordinarily complex. Natural languages, like English, are context sensitive languages and require a great deal of additional information to interpret them – information which is not necessarily available through the text by itself.

3. Semantic: This is not so much a third category as it is a higher level of analysis. This level requires that lexical or syntactic analysis (or both) have been performed (in some fashion). At this level we gain access to the meaning of a document. This is the most complex form of analysis of the document, but it is potentially the most fruitful. By gaining an understanding of the meaning of a document, the SA has the ability to perform deeper comparisons and possibly to learn from deeper patterns of information to perform better as an SA. The algorithm addressed in this work is not a learning algorithm, but there is a lot of potential for extension by looking at the supervised and unsupervised learning aspects of this problem. It is possible to get at semantic information via lexical and statistical analysis, but there is room for debate as to what the meaning of this would be (since this would be treating natural language as regular for this context).[40]

Of the above kinds of comparisons, the proof of concept employed in this thesis is primarily applying the lexical and statistical analysis strategy, with some variation on this. This

[39] Hill-Climbing can sometimes get stuck at local maxima and plateaus, without being able to move away from these. I don't think that the WWW would present this problem for the information retrieval domain, but it is possible that a Hill-Climbing solution by itself would not be able to break free of a local document cluster to find more relevant information somewhere else.
[40] [Chomsky, 1957, 1968, pgs. 18-19]

approach is a simplified method of interpretation and does not succeed in a deep interpretation of the text. It is important to describe the general assumptions related to relevance in this work and the approach used.[41] To simplify the analysis, words used in a document can be broken down into two basic sets; the set of words which are related to the content of the material and the set of words which act primarily as tools of syntax. We can call these sets C and O. Where C is the set of content indicators and O is the set of syntax operators. Since these sets are context sensitive, they will necessarily overlap. This work deals with documents within the Universe of Discourse or U which is defined by the set of all legal tokens in the English language. In this context a sentence can be viewed as an ordered set of tokens and this set is produced from the union of O and C mapping to the set of ordered sets (or n-tuples) which are pragmatically acceptable (even if not grammatical) sentences.[42] Finally, from the ordered set of tokens in a given document there are many mappings to another set M. M is a set of all such mappings.

Given the description of the problem up to this point, very little can really be said about a document, especially if we intend to avoid the thornier issues of syntax and semantics. That a document is an ordered set of legal tokens is clear, and certainly this can indicate something about the placement of a document. This relationship between similar documents is something that can be defined, and given definitions of varying resolution. This work contends that there exists a set of mappings from any one document (ordered set of tokens) to another set and these mappings can have differing discriminatory power. The sets themselves vary in the strength of their relationship to any document, and therefore should vary in the weight or value assigned for classifying a document.

Listed below are some of these mappings:

1. $f_1 : S \rightarrow WL$, where S is the sample document and WL is the set of ordered pairs (a,b), such that a is a word in S and b is its relative frequency. This is the most basic lexical comparison between documents.

2. $f_2 : S \rightarrow WC$, where S is the sample document and WC is a set of ordered pairs (a,b), such that a is a content related token ($a \in C$) from S and b is the relative frequency of this token.

3. $f_3 : S \rightarrow TC$, where S is same as above and TC is a set of ordered pairs (a,b), such that a is $a \in O$ and b is the relative frequency of a.

[41] [Burnard, 1989, (internet document)] **This source is weak but I think it stresses some key points related to relevancy and textual analysis. Most important is that even though the very basic and naïve approaches (as with the approach I have taken) are easier to implement, we should not limit ourselves to variations on statistical analysis.
[42] Pragmatics is a field of language theory which is concerned with the actual use of language versus the ideal use of language. In reality, most native users apply rules to the daily use of language which do not fit neatly into a standard grammar.

4. $f_4 : S \rightarrow OP$, where S is same as above and OP is a set of 3-tuples (a,b,c), such that a is is S, b is in S, b is 1 place ahead in the ordering than a or b = a + 1. And c is the relative frequency of this pair of words.

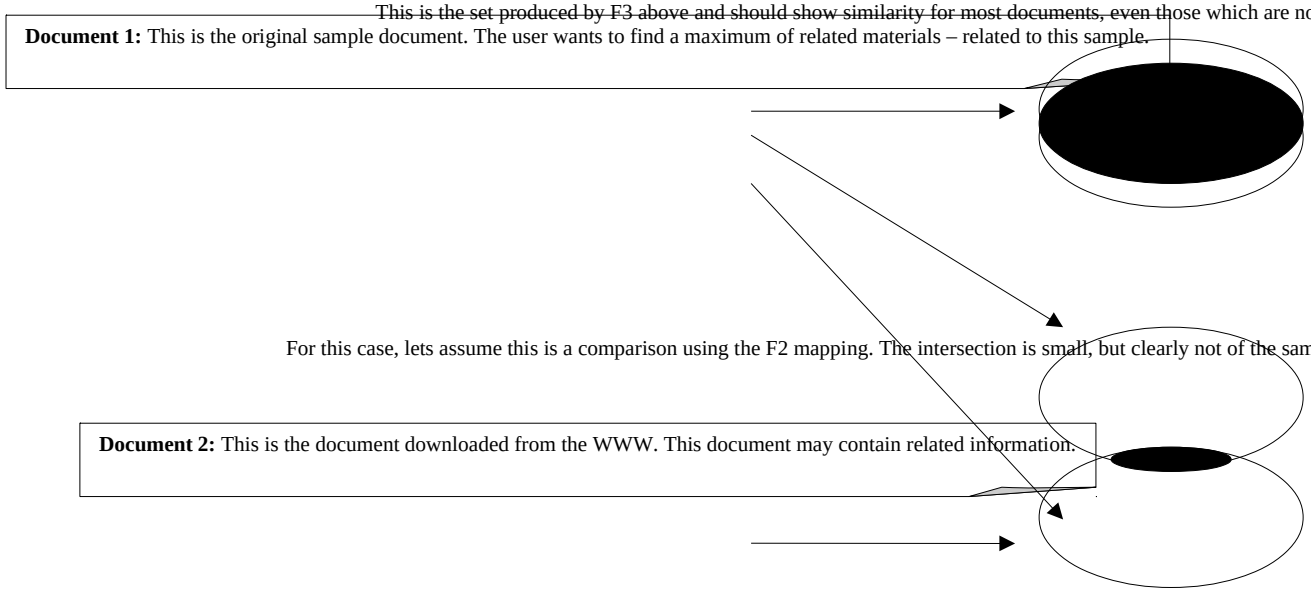
5. $f_5 : S \rightarrow MXST$, where S is same as above and MXST represents a set of 3-tuples, which is a subset of OP and represents the maximum spanning tree connecting all words in the document based upon their ordering.

Of the comparisons listed above, f_2 seems to be the most promising for narrowing a search at the most basic level and focusing solely on what are apparently content words. The problem with this approach is that it may not be sufficiently absorptive to include all of the ambiguous lexical relationships which can indicate similarity. The key word here is 'similarity', not sameness. The function which will compare two documents needs to return a value indicating similarity for placement in a corpus. However, one of the contentions of this work is that the number of mappings used and the weight given to each is related to the accuracy of the relevance evaluation.[43]

Additionally, how do we know which mapping is the best? Some are more resource intensive than others.[44] And, some will indicate similarity much better than others. Of the mappings identified above, clearly f_3 will be weakest at showing similarity between documents. Certainly, we will want to weight mappings differently based upon their presumed power of showing similarity between documents. (See Figure 2 below)

[43] A corpus is a set of documents, and usually similar documents.
[44] I am speaking primarily of the order of magnitude for the function algorithmically.

Figure 2: Sets which show similarity (some more strongly th



f_5 is intriguing because it attempts to identify potentially syntactic and semantic related aspects of the document but it does not require any grammar to be identified. It operates on the extension of f_4 and claims some meaningful value to knowing the most strongly related subset of the ordered set OP. With respect to f_5, there is an implementation in C++. This implementation is for both a graph representation of a text document and a maximum spanning tree function (this is based upon Kruskal's algorithm).

Each one of these mappings can be generated, and with the exception of f_4 and f_5 these mappings are not terribly resource intensive to implement in a computer program. But, as stated above, we can say two things of them all; there is no way of knowing if a mapping is not relevant absolutely, and some mappings are better at identification than others and should be weighted differently.

Given this circumstance, it is not in the scope of this work to identify all useful mappings but to propose the following for this relevance function. That relevance is a function of all mappings from two documents X and Y multiplied by their individual weight value (wv) for identification, and that the sum of these weights should be less than or equal to 1. This value represents the relationship between X and Y where X is the original document and Y is the new document for comparison. More formally:

$$R(X,Y) = (1(X,Y)*wv1) + (2(X,Y)*wv2) + (3(X,Y)*wv3) + \ldots + (n(X,Y)*wvn)$$

As an example, if $f_1 = .25$, $wv_1 = .5$, $f_2 = .35$, $wv_2 = .2$, and $f_3 = .65$, $wv_3 = .3$, then R(X,Y) would be 0.327 or roughly 33% relevancy. The complexity of this solution is not simply in determining what and how many mappings of this type there are, but also there is a basic requirement of knowing how relevant each mapping is to determining similarity between documents.

Researchers, over the past few decades, have stepped away from ideal approaches to natural language processing to adopt more pragmatic strategies. Although this relevance function is fairly primitive, it was developed with the idea of solving a problem as simply as possible, without having to develop or include more complex methods. Also, embedded in this conception is a notion that there are probably an infinite number of mappings from a sample document to some other congruent set – some of these may include syntax parsing. Basically, I have looked at some fairly simple mappings.[45]

There is no claim to a basis in or an extension to linguistic theory. This is important to stress, since the types of ordered sets being discussed are samples of natural language. This may not be an insurmountable weakness, and it is beyond the scope of this work, but to ignore the specialist knowledge when filtering information can introduce errors and misinterpretations.[46]

The relevance function discussed in this section is very basic. It would have been interesting to spend more time researching just the problem of relevance in the field of information retrieval. There is a lot of work to be done investigating the application of neural networks to determining the similarity between documents and document sets. If a good description of the feature set can be determined, then there may be much more effective ways of doing relevancy using trained neural networks.

[45] [Maedche, 2002, pg. 97]
[46] [Navarro and Raffinot, 2002, pgs. 2-3]

4.0 – Results

The information included in this portion of the work is not conclusive. It represents one run of the PERL implementation attached to this paper and a multiple run of the graph algorithm with comparison data generated in MS Access. In section 5 this work will review the results and propose interpretation.

4.1 – A* Run Data

These results are based upon one test of the PERL program over a period of 1 hour approximately. To fully test, de-bug and improve this method could take much longer and is beyond the time frame available. But is would be important, in order to draw any conclusions, to extend this testing and to improve the method itself. To conduct a test of this program this thesis paper was used as the primary test sample. Here are the summary results:

Criteria = 10000 seconds

Relevancy Constraint = 10%

Gamma = 0.5

Max Bandwidth = 10 kilobytes per second

BPA (Best Path Adjusted) = 18 megabytes of relevant data

Total Data Collected = 1.5 megabytes

Links Collected = 12413

Links Visited = 122

Different adjustments to the weightings of each of the 4 mappings were used. These mappings are those discussed in section 3.3, with the exception of the maximum spanning tree (see the next section for this). It is not clear if further experimentation with these weights will change performance, but there was some experimentation with different weights and the ordered pair mapping by itself did not perform well as a relevance measure in most of the tests. There is reason to believe that reinforcement learning could be used to adjust these weights. This has not been explored, but it certainly represents an interesting extension to the work.[47]

Below are detailed results of the A* Test performed:[48]

LINKS	URL SIZE	DATA SAVED	A*	REL	f_1	f_2	f_3	f_4	TIME
www.hut.fi/Misc/Electronics/docs/audio/spdif.html	28676	28676	1496.4	0.42	95.51	25.66	91.72	5	0
www.nlta.nf.ca/publisher/help/5search.htm	19857	48533	2002.3	0.45	94.74	33.33	88.76	5	9
help.yahoo.com/help/us/ysearch/ysearch-03.html	907	49440	41272.3	0.48	77.36	50.00	65.01	5	15
www.syntest.com.tw/faq/turbofaultfaq.html	13672	63112	3178.6	0.42	96.10	24.63	90.68	5	19
www.isrl.uiuc.edu/dubinabs.html	10766	73878	3682.0	0.46	96.34	32.71	89.47	5	25
www.cs.uwaterloo.ca/cs-archive/CS-2001/CS-2001.shtml	8888	82766	4286.3	0.47	91.53	38.23	90.53	5	33
www.uiah.fi/projects/metodi/140.htm	382	83148	269743.2	0.17	68.58	4.35	5.00	5	40
www.geocities.com/bayinnaung/progexampweblog.html	1869	85017	21955.7	0.44	88.07	33.61	83.37	5	46
www.data-man.com/nwltr12.html	3688	88705	10549.4	0.46	93.19	35.98	86.53	5	50
www.dlib.vt.edu/Papers/SIGIR96/SIGIR96.Env.html	34790	123495	1216.7	0.45	97.45	29.36	93.71	5	56

[47] In section 4.1 there is a complete table of values for this experimental search.

[48] I used my thesis as the seed document for this test. The averages are shown at the end of the table. For the value in column F4, this function is not fully debugged and will return a value of 5 under conditions where the relevancy is too low.

galeb.etf.bg.ac.yu/~vm/tutorial/internet/business/ebi2/ebi4.html	28773	152268	1530.6	0.43	96.21	26.61	90.69	5	63
www.cs.usyd.edu.au/~bob/um96-paper.html	7338	159606	5027.6	0.49	89.79	42.12	91.20	5	75
citeseer.nj.nec.com/chau03design.html	6266	165872	6927.9	0.42	93.24	30.16	68.63	5	80
best.me.berkeley.edu/~adong/cad-sys.html	9554	175426	4013.2	0.48	94.70	38.15	85.30	5	85
dollar.biz.uiowa.edu/~fil/IS/info-spiders.html	15221	190647	2614.8	0.46	95.37	34.45	91.19	5	89
wwwis.win.tue.nl/~debra/infwet99/aroyo.html	15219	205866	2606.9	0.47	95.22	36.57	88.22	5	122
www.sics.se/diglib/delos8ws/abstracts.html	7030	212896	5792.5	0.45	94.55	32.42	85.80	5	127
www.cs.jhu.edu/~weiss/papers.html	8915	221811	4481.7	0.46	93.94	35.19	85.14	5	132
vcolaso.port5.com/ai/vicdoc/Genetic%20Algorithm%20for%20Internet%20Search.htm	8682	230493	4287.7	0.49	96.49	40.52	88.23	5	137
www.bradleyrhodes.com/Papers/remembrance.html	19649	250142	2105.7	0.46	95.64	33.20	94.37	5	143
www.resna.org/taproject/library/conprov.html	4156	254298	9952.5	0.44	96.00	30.14	83.19	5	152
naweb.unb.ca/proceedings/1997/tsinakos/tsinakos.html	14540	268838	2662.5	0.49	95.60	39.16	90.87	5	157
www.searchtools.com/info/intranets.html	15700	284538	2726.5	0.44	97.25	29.28	90.45	5	163
citeseer.nj.nec.com/craswell01effective.html	4424	288962	9928.1	0.41	91.73	30.04	66.83	5	170
stanford.edu/group/partners/skillset.shtml	1454	290416	28619.6	0.43	82.39	37.76	64.77	5	176
www.stanford.edu/group/partners/skillset.shtml	1454	291870	28624.3	0.43	82.39	37.76	64.77	5	183
www.public.asu.edu/~ehoran/ENG500F02.htm	4270	296140	10230.3	0.41	93.34	27.43	79.62	5	197
tools.com/info/intranets.html	73	296213	1940352.0	0.13	5.00	20.00	5.00	5	202
www.ladseb.pd.cnr.it/infor/ontology/Papers/Ontobiblio/ComputerScienceP.html	12526	308739	3808.7	0.39	93.41	29.82	47.55	5	211
www.kcl.ac.uk/neuronet/about/roadmap/imsystems.html	1371	310110	28235.8	0.46	85.36	41.24	73.78	5	219

www.parc.xerox.com/istl/groups/did/didpublications.shtml	33774	343884	1516.3	0.41	96.87	23.29	91.96	5	227
www.cordis.lu/ist/ka3/iaf/iaf_workshop.htm	11845	355729	3573.6	0.45	95.27	31.65	90.46	5	248
www.ntu.edu.sg/sce/Syllabus99.htm	19874	375603	2444.1	0.41	98.02	25.58	74.71	5	258
www.cc.gatech.edu/gvu/nsf-ws/report/Indexing.html	11446	387049	3517.2	0.48	95.11	36.84	91.35	5	274
sarasavi.cmb.ac.lk/academic/Science/Computer/dscs/courses/msc_units_details_new.htm	14046	401095	3367.8	0.41	97.91	27.07	66.59	5	283
www.computer.org/proceedings/HICS29/VOL2/TOC.htm	4780	405875	10462.1	0.36	82.62	31.10	31.66	5	295
www.state.la.us/osp/AgencyCenter/FAQs-agcy.htm	6349	412224	6844.5	0.42	93.15	28.04	87.87	5	305
www.nsac.ns.ca/lib/instruct/6internt.htm	13554	425778	3130.8	0.46	96.59	33.55	91.66	5	317
www.onlinemag.net/OL1995/NovOL95/notess.html	11376	437154	3724.5	0.46	95.51	32.63	91.70	5	328
www.rlg.org/shares/arifax.html	15398	452552	2976.5	0.43	94.18	30.17	84.84	5	338
library.bowdoin.edu/services/docdel.html	4616	457168	8474.9	0.47	96.37	36.32	84.21	5	353
www.nap.edu/books/0309082749/html/407.html	17653	474821	2747.1	0.42	93.69	26.23	91.25	5	364
www.gov.sk.ca/bureau.stats/docs/costrec.htm	23590	498411	2139.5	0.43	97.45	25.45	93.37	5	382
www.dell.com/us/en/biz/topics/power_ps3q00-beowulf.htm	12124	510535	3414.4	0.48	96.37	38.10	87.24	5	396
discovernd.com/itd/planning/qa99.htm	6084	516619	6670.6	0.46	91.37	35.74	88.89	5	412
depts.washington.edu/gcs/gc1hint.html	1536	518155	27465.6	0.42	81.23	37.23	62.51	5	428
www.housing.act.gov.au/Policy/RentPayment.htm	4616	522771	9901.8	0.40	85.90	26.88	84.58	5	441
www.infotoday.com/online/OL1999/toth7.html	14484	537255	3017.3	0.47	94.16	36.42	92.88	5	456
ai.ijs.si/mezi/agents/agents.html	9817	547072	4495.2	0.44	92.00	32.16	88.66	5	470

www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-AAAI/Letizia.html	19182	566254	2445.9	0.47	94.63	35.16	93.45	5	486
www.cse.unsw.edu.au/school/publications/2000/SCSE_publications.html	51	566305	1243576.2	0.28	5.00	50.00	5.00	5	506
www.lcr.thomson-csf.fr/projects/planet/ws1_work.html	15786	582091	3050.5	0.44	94.13	30.44	90.79	5	532
www.cs.mu.oz.au/pgrad/seminars/1998.html	10986	593077	4045.2	0.46	91.46	34.07	92.29	5	550
rti7020.etf.bg.ac.yu/rti/ebi/dipl/dragana/RAD1.htm	22117	615194	2263.0	0.47	96.08	34.82	92.74	5	567
www.geometry.net/detail/calculus/limits_and_continuity_page_no_4.html	4076	619270	10356.7	0.44	83.45	34.78	86.60	5	587
www.nap.edu/books/0309051932/html/25.html	27388	646658	2117.0	0.43	94.83	26.58	93.25	5	608
www.at.vcu.edu/sas/vieditor.html	16434	663092	2949.4	0.46	95.01	33.64	90.96	5	630
bi.snu.ac.kr/~scai/Publications/pub.html	93	663185	2938157.8	0.06	5.00	7.69	5.00	5	652
www.nal.usda.gov/afsic/AFSIC_pubs/qb9709.htm	106042	769227	1270.5	0.32	96.68	9.42	73.57	5	672
algo.inria.fr/seminars/sem96-97/flajolet.html	11438	780665	4263.6	0.43	89.79	31.12	88.35	5	769
www.cmis.csiro.au/Library/Publications/pubs1997.htm	0	780665	##########	0.05	5.00	5.00	5.00	5	791
www.cdam.lse.ac.uk/BCB/partC2001.html	31148	811813	2420.7	0.35	92.74	17.13	67.99	5	816
www.cs.brown.edu/memex/index.html	20551	832364	3169.0	0.36	96.82	15.21	83.84	5	840
www.di.unipi.it/~gulli/researches/hp-grants/proposal97.htm	19765	852129	2873.5	0.44	96.56	29.01	91.69	5	871
ai.bpa.arizona.edu/papers/cscw94/cscw94.html	44025	896154	1792.5	0.45	97.63	30.63	92.56	5	901
www.cs.dal.ca/~eem/res/ThesisTopics.html	19324	915478	2827.0	0.47	96.62	35.65	92.31	5	931
www.it.bton.ac.uk/staff/vl9/paper/paper.html	19601	935079	2795.1	0.48	94.46	38.25	92.93	5	961
www.dimi.uniud.it/~ift/aiia/html/AI_IA.html	32887	967966	2161.4	0.46	97.76	32.36	89.65	5	993
wwwis.win.tue.nl/infwet97/proceedings/resource-limited-full.htm	17538	985504	3060.4	0.49	95.10	38.28	94.12	5	1028
www.switch.ch/edu/research_index.html	0	985504	##########	0.05	5.00	5.00	5.00	5	1064

www.research.ibm.com/journal/sj/413/perrone.html	3299	988803	11531.3	0.50	90.21	45.03	80.62	5	1098
www.cs.utexas.edu/users/dmg/seminar/abstracts.html	18926	1007729	3176.8	0.45	97.01	30.77	91.09	5	1140
www.jrc.es/pages/iptsreport/vol68/english/ICT2E686.html	15640	1023369	3501.5	0.47	97.41	35.80	91.09	5	1177
www.cra.org/Activities/craw/dmp/awards/lefevre_web/applications.html	7730	1031099	5692.9	0.50	94.17	41.26	90.44	5	1219
www.bilaney.com/webpages/ris_pages/rmtech.htm	26210	1057309	2784.5	0.44	95.28	29.07	90.42	5	1259
www.weblogic.com/docs51/admindocs/http.html	356	1057665	152955.1	0.31	66.13	33.33	5.00	5	1303
www.cisco.com/en/US/tech/tk652/tk701/technologies_tech_note09186a00801006c6.shtml	14615	1072280	3923.4	0.46	90.06	35.28	92.84	5	1353
www.duckware.com/bugfree/chapter6.html	23163	1095443	3038.3	0.46	92.00	35.27	92.32	5	1402
www.his.com/~z/ftirp.html	30927	1126370	2741.9	0.44	93.85	30.10	94.33	5	1461
www.loristech.com/FAQs.html	22154	1148524	3302.0	0.44	95.57	28.48	93.09	5	1510
www.w3.org/TR/REC-html40/appendix/notes.html	30767	1179291	2841.5	0.45	96.08	29.88	93.83	5	1560
members.nccw.net/fmacall/readme.htm	31331	1210622	2931.4	0.42	95.94	25.88	92.88	5	1614
www.cs.unc.edu/Research/ProjectIndex/AreaDescriptions.html	220	1210842	277707.7	0.28	47.42	33.33	5.00	5	1670
cs.anu.edu.au/honours/topics.html	9629	1220471	5508.1	0.47	93.17	35.89	92.41	5	1723
www.ariatg.com/research.htm	220	1220691	260007.4	0.30	48.41	33.33	21.95	5	1786
wp.netscape.com/compass/v3.0/evalguide/advantages.html	0	1220691	##########	0.05	5.00	5.00	5.00	5	1841
www.thunderstone.com/texis/site/pages/texisdetail.html	44922	1265613	2847.0	0.43	94.03	27.24	92.01	5	1897
theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html	16461	1282074	4109.4	0.49	94.40	39.63	92.87	5	1973
marylaine.com/exlibris/xlib18.html	748	1282822	57015.3	0.41	78.07	35.48	63.41	5	2036

dimacs.rutgers.edu/Workshops/Codes/abstracts.html	15222	1298044	4636.8	0.44	96.94	29.60	92.18	5	2100
cobrands.smallbiz.findlaw.com/business_commercial/legal/source/library/lg_environment/articles/ph000013.html	23721	1321765	3853.3	0.44	96.37	27.84	93.60	5	2166
www.nap.edu/books/0309044839/html/55.html	16510	1338275	4712.0	0.42	97.34	24.69	92.18	5	2239
www.as.wvu.edu/~tmiles/assign2new.html	9564	1347839	6195.0	0.46	95.77	32.80	91.09	5	2311
us/policies/p5310.html	483	1348322	96002.0	0.37	49.01	40.00	60.52	5	2383
wvde.state.wv.us/policies/p5310.html	33247	1381569	3732.1	0.42	95.17	26.62	86.95	5	2457
www.khsd.k12.ca.us/district/technology_resource/level1/Default.htm	3340	1384909	14618.6	0.41	82.88	32.86	74.79	5	2543
www.jamia.org/misc/ifora.shtml	40887	1425796	3662.1	0.43	96.61	26.14	93.15	5	2627
www.acm.org/sigchi/chi95/Electronic/documnts/papers/sm_bdy.htm	23200	1448996	4286.6	0.48	96.61	37.17	92.44	5	2719
ing/debra/article.html	92	1449088	657081.2	0.28	5.00	50.00	5.00	5	2805
archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/debra/article.html	24455	1473543	4450.2	0.46	92.07	34.73	93.45	5	2896
www.cse.unsw.edu.au/~mbarg/prdex.html	9109	1482652	6644.2	0.51	95.60	43.16	91.59	5	2985
nlp.shef.ac.uk/research/students/research.html	8029	1490681	7441.0	0.48	96.11	37.92	90.08	5	3079
www.geog.port.ac.uk/gbhgis/jtap_ehap/soc_cart_article.htm	27668	1518349	4654.0	0.43	95.71	27.20	93.91	5	3174
www.doc.ic.ac.uk/~srueger/pr-s.sewraz-1999/abstract.html	15540	1533889	5665.6	0.46	95.50	33.50	94.10	5	3276
www.library.ucsb.edu/untangle/lager.html	24022	1557911	4965.3	0.46	96.70	32.93	92.42	5	3375
www.nrc.gov/reading-rm/doc-collections/enforcement/actions/materials/ea02045.html	10802	1568713	7053.1	0.44	96.48	29.81	86.61	5	3479
icpr2002.gel.ulaval.ca/tutorialsForWWWPage.htm	19593	1588306	5606.0	0.44	97.17	30.47	87.67	5	3612
Averages:	**14843**	**727567**	**47675873460.4**	**0.42**	**86.03**	**31.37**	**77.72**	**5**	**979.01**

4.2 – Maximum Spanning Tree Mapping Data

A graph algorithm using an adjacency list implementation provides the basis for the experimentation with the Maximum Spanning Tree Algorithm. For more detail on the purpose and use of this, review sections 5 and 6. This is, however, an implementation of one of the mapping discussed in section 3 in the context of document relevance criteria. The implementation is in C++ and was run successfully using the GNU compiler on our Solaris cluster here at IUPUI. The results of experimentation are interesting, and warrant use as an additional feature for comparison. The experimental data below resulted from a single test with 3 documents; all of which were news articles downloaded from AP. 2 of these articles dealt with the conflict in Iraq, one of these articles dealt with the flashpoint on the Korean peninsula:

1. For a document sample [A] with 513 3-tuples containing node, edge and weight, a document [B] in the same subject area had 59 pairs which matched completely.

2. A document [C] outside of the subject area had a match of 9.

If a properly designed matching function were built, there would be a higher resolution comparison and would show greater sensitivity to partial matches (matches where node and edge are the same but relative frequencies vary). The current test is exact match of pairs and frequency, but it would not be difficult to accept pair matches and apply scores based upon the difference between the relative frequencies of the pairs (with respect to the original document).

In the table below is a comparison of perfect maximum spanning tree matches (matches where node, edge and frequency are the same). The comparison operator would be more effective, as stated, if this function processed the relative frequencies as degrees of difference versus total equivalence.

Original: Iraq Content	New 1: Iraq Content	New 2: Korea Content
WAR 1 WITH	WAR 1 WITH	
WAR 1 REUTERS	WAR 1 REUTERS	
TORTURE 1 CENTER	TORTURE 1 CENTER	
TO 1 PROBE	TO 1 PROBE	
THE 2 CAPITAL	THE 2 CAPITAL	
THE 1 STRONGEST	THE 1 STRONGEST	
THE 1 STATEMENT	THE 1 STATEMENT	
THE 1 REPORT	THE 1 REPORT	THE 1 REPORT
THE 1 OFFICE	THE 1 OFFICE	
THE 1 COUNTRY	THE 1 COUNTRY	
THAT 1 SERVES	THAT 1 SERVES	
TERROR 1 PLOTS	TERROR 1 PLOTS	
STRONGEST 1 BLASTS	STRONGEST 1 BLASTS	
SPECIAL 1 COVERAGE	SPECIAL 1 COVERAGE	

SAID 1 PEOPLE	SAID 1 PEOPLE	
SAEED 1 ALSAHHAF	SAEED 1 ALSAHHAF	
SADDAMS 1 SON	SADDAMS 1 SON	
SADDAM 1 HUSSEIN	SADDAM 1 HUSSEIN	
RUMOR 1 CIRCULATED	RUMOR 1 CIRCULATED	
PRESS 1 WRITER	PRESS 1 WRITER	PRESS 1 WRITER
PALACE 1 PRESIDENTIAL	PALACE 1 PRESIDENTIAL	
ONLY 1 MINOR	ONLY 1 MINOR	
OLYMPIC 1 COMMITTEE	OLYMPIC 1 COMMITTEE	
OLD 1 PALACE	OLD 1 PALACE	
OF 1 DETERMINING	OF 1 DETERMINING	
NO 1 WAY	NO 1 WAY	
NEWS 3 WEB	NEWS 3 WEB	
NEWS 1 JORDAN	NEWS 1 JORDAN	
MOHAMMED 1 SAEED	MOHAMMED 1 SAEED	
MINUTES 4 AGO	MINUTES 4 AGO	
MINISTER 1 MOHAMMED	MINISTER 1 MOHAMMED	
JORDAN 1 FOILS	JORDAN 1 FOILS	
IS 1 LINKED	IS 1 LINKED	
IRAQI 2 TV	IRAQI 2 TV	
IRAQI 1 NATIONAL	IRAQI 1 NATIONAL	
INTERACTIVE 1 DOWNTOWN	INTERACTIVE 1 DOWNTOWN	
INFORMATION 1 MINISTER	INFORMATION 1 MINISTER	
HAVE 1 FOCUSED	HAVE 1 FOCUSED	
HAD 1 HOPED	HAD 1 HOPED	
FOILS 1 TWO	FOILS 1 TWO	
DEFENSE 1 DEPARTMENT	DEFENSE 1 DEPARTMENT	
BEGAN 1 MARCH	BEGAN 1 MARCH	
AT 1 LEAST	AT 1 LEAST	
ASSOCIATED 1 PRESS	ASSOCIATED 1 PRESS	ASSOCIATED 1 PRESS
ARAB 1 ANGER	ARAB 1 ANGER	
AND 1 QUSAI	AND 1 QUSAI	
AND 1 HIS	AND 1 HIS	
AGO 1 SPECIAL	AGO 1 SPECIAL	
A 1 TORTURE	A 1 TORTURE	
A 1 RUMOR	A 1 RUMOR	
A 1 COMPLEX	A 1 COMPLEX	
<START> 1 IRAQI	<START> 1 IRAQI	
<START> 1 INTERACTIVE	<START> 1 INTERACTIVE	
<START> 1 DEFENSE	<START> 1 DEFENSE	
<START> 1 COALITION	<START> 1 COALITION	
<START> 1 AP	<START> 1 AP	
<COMMA> 2 WHERE	<COMMA> 2 WHERE	
<COMMA> 1 WHICH	<COMMA> 1 WHICH	<COMMA> 1 WHICH
<COMMA> 1 ASSOCIATED	<COMMA> 1 ASSOCIATED	<COMMA> 1 ASSOCIATED

WEB 3 SITES		WEB 3 SITES
IRAQ 1 NEWS		IRAQ 1 NEWS
<START> 2 BUT		<START> 2 BUT
<START> 1 OFFICIALS		<START> 1 OFFICIALS
Total	Total	Total
513	59	9

5.0 – Discussion and Conclusions

The investigation of the problem domain, as described so far, reveals how complex a task it is to filter, analyze and place documents in a particular category. The A* method used functions correctly as a PERL application, but its performance indicates that further refinements are necessary for any practical implementation. The techniques employed to build the SA in PERL were traditional and some of the program elements are not optimal.[49] Because of the experience level of the designer/programmer and the complexity of the domain, it was important to be as conservative as possible with respect to implementation of the method.[50]

5.1 – Observations

The initial task of this investigation was to be the exploration of topological ordering to the domain of information filtering. This use of topology seems warranted, but there was not enough time to effectively define and implement a coherent method using only topological techniques. The example of the spanning tree algorithm given in this work is intended to portray

[49] In one case the PERL Array, storing a links object, was used to imitate a priority queue. This is not the best way to implement this data structure, but it was a quick way to complete the program with the time allowed. Basically, PERL was still being learned as the program was being written.
[50] [Russell and Norvig, 1995, pgs. 96-99]

some of the insights gained during the first few months of research – the use of the maximum spanning tree algorithm demonstrates some interesting possibilities. The implementation of the spanning tree method, in this work, is still very preliminary. It is important to note that any graph implementation of text document ordering can have significant memory requirements.[51]

The mapping functions which were used to test the methodology worked with varying degrees of success. The f_4 mapping has particular problems, and will return a default value of 5% for certain error conditions. The problems demonstrated in the results section may be due, in part, to the implementation. However, some of the issues relate back to the usefulness of some mappings for determination of document relevance.

The design of the proof of concept for this A* method could have been more object oriented and the use of PERL as the tool for implementation may not have been correct. The program, as written, is modular and takes advantage of the PERL syntax for packages. The basic descriptions of necessary components, as shown in the experimental section, are sufficient to begin the process of developing an SA in PERL, but more design resolution is necessary. Also, some of the file outputs of this program were not used effectively, because the time for further development ran out. There is one file, labeled ontology, which contains a raw listing of ordered pairs after the operator-token filter has been applied. This ontology file was intended to be the basis for a topic map and could still be used as a topic map for the organization of links removed from the HTML documents.

Finally, the sample set for the testing of this method is too small for valid statistical inference. Therefore, the results are intended to provide only possible indications of the behavior of this A* method and are not provided as proof (pro or con) for the effectiveness of the method.

5.2 – Lessons Learned

As with any adventure in learning, after the arrival at the destination there is time to review the good and bad points of the journey. The following is a list of actions which could have been taken and which seem to be better approaches in hindsight:

1. Further narrow the topic to one aspect of determining relevance. Instead of looking at several different kinds of mappings, one specific mapping (feature) should have been investigated thoroughly – like the spanning tree for the document – instead of trying to deal with the wider issue of how these features interact.

2. Investigate what both fuzzy systems theory and neural networks have to say concerning this topic. There would have been more work involved, but the relevancy function might

[51] Although the use of adjacency lists, as in my implementation, may reduce memory costs it is still very expensive to represent a topology in computer memory. The most common means of representation is an adjacency matrix, and in this case the cost to memory is virtually guaranteed to be n^2 – where n is the number of nodes in the graph.

be more effectively implemented as a neural network. Because of the complexity and work involved, this would have had to become the principal focus of this senior thesis.

3. Determine early what tool would be best for experimenting with these problems. There exist fully developed tools, components and classes for accomplishing many of the tasks described in this work. Much time was spent investigating different tools for portions of the problem, but this extra work did not necessarily result in any additional methods.[52]

4. One important weakness of this implementation is that the relevance function is not complete and the ability to make better guesses about the significance of links contained in a page is not developed at all. A better solution needs to include effective heuristics for determining if a link on a web page is actually related to the subject matter, and to be able to do this without visiting (downloading) the page. This would be very difficult, but human search agents (as we all are) make these distinctions as part of their process and the effectively avoid advertising links which are not related to the information being sought. The program, as written, does not make this distinction.

The implementation of A* in this work does appear to perform adequately and does retrieve genuinely relevant materials. This investigation looked at many problems, possibly too many. Overall this project has led to a greater appreciation of why it is that companies, like Google, require so much intellectual capital to perform their function and provide service to their clients.

It is important that information scientists and workers deal with the issues of information explosion effectively. Some of the solutions may be procedural in nature and may involve the adoption of better standards for indexing and identifying materials proactively, although this is not nearly the whole solution. There is general agreement that along with better methods for markup and indexing, a need remains for more intelligent systems to assist humans with the search for information.

5.3 – Future Work

Below is a listing of proposed future work, which is of interest, and provide a basis for many other fruitful investigations:

1. ***Hill-Climbing and Simulated Annealing:*** I think a hill climbing solution would show greater promise and be far simpler to implement. If I decide to explore this problem domain further I think I will explore the use of the hill climbing algorithm for heuristic evaluation.[53]

[52] I spent about 3 months looking into PROLOG as an intermediate representation of the content of a document. I thought that this semantic information could be produced – with low resolution – simply through parsing the natural language and using a simplified grammar (and a lossy one) for doing this. I still think this would be interesting to explore, especially with reference to machine learning, but I don't think this was a reasonable problem to take on for this thesis.

[53] [Mock, 1994, pgs. 1-3]

2. ***Modeling User Profile:*** There is nothing in this work directly related to modeling user profiles, but it would be important for an SA to gain an understanding of user interests in a non-intrusive fashion. This is also where applying reinforcement learning models may help.

3. ***Natural Language Processing:*** I would like to explore the development of semantic mappings based upon the parsing and interpretation of natural language. There was not enough time available to explore this, but I think it would have been another key factor for testing similarity. I did experiment a little with the parsing of sentences, as a context free grammar, and converting this information into a predicate form – this work became too complex and infeasible given the time constraints.

4. ***Applying Neural Networks to Relevancy:*** This has been mentioned previously in the paper, but it is clear to me that neural networks could be an effective means of dealing with information relevancy. Clearly you could train a neural network to recognize documents that belong in a corpus or are rejected.

5. ***Topological Ordering for Ontology Building:*** I did generate ordered pair listings from content data. I think this information could be used to generate naïve ontology. This could be used as an automated way of organizing materials gathered from a file space which could assist in information management.[54]

6. ***Designing Information Structures which are Self-Organizing:*** I am very interested in extending the object oriented paradigm to the modeling of independent and goal oriented data-structures. Data structures which have a real sense of identity and are capable of generating inheritance relationships at run time to produce new data-structures. I see this as a generative approach to data management and exploration, and possibly as the basis of a model for intelligent system memory. This was clearly beyond the scope of this work. There may be some correlation between this and how memory is ordered in the human mind.[55]

7. ***Benefits of the WWW as an AI Problem Space for Testing Concepts:*** I hope in any of my future explorations I am able to leverage the WWW as a problem space. I think this space is unique because of its nearly equalizing power. In all other arenas, human perceptual modalities far surpass computers. However, sense the web is fundamentally a data space; the principal token of perception in this space is binary and fully accessible to machines. So, in this space I think the playing field is more even and it enables a more interesting exploration of intelligent systems.

8. ***Improved Browser Technology:*** I think the future of information retrieval from the WWW will be more dependent upon smarter technologies than upon better markup languages. This is a conjecture, but I think systems like the one I have described (certainly more advanced) could be the basis of browsers fully utilizing the resources of the internet and computers without interfering with the interests and needs of the user. Any use of interface agents should be collaborative in nature.[56]

5.4 – Conclusion

[54] [Sowa, 2001, (internet document) and Woods, 1995, (internet document)]
[55] [Hardy, 1998, pgs. 7,12,14]
[56] [Lieberman, Fry and Weitzman, 2000 (internet document)]

This project began with ambitious goals from the frame of reference of the investigator. However, as a senior thesis it was not intended to be the exploration of fundamental work and was given explicit time restrictions. If this were a sample of a graduate project/thesis, then there would clearly be much more work ahead and probably more narrowing of the topic. As an undergraduate thesis, this work was indispensable in my development as a critical thinker and as an information scientist.

What I would like most to do, if given the opportunity, is teach a class to undergraduate students which deals solely with this topic. I believe many of the more interesting and fundamental concepts of computer and information science can be covered in a course which uses the search for information as its vehicle. Actually, one of the most basic forces behind our interest (as humans) in information science is our need to learn more and to remember what we have learned – searching and classification. Maybe, I search therefore I am?

6.0 - Experimental

The hybrid A* search could be embedded inside a web browser. As an SA, it could monitor the information content of the user's current search and then, when it appears the user is idle, conduct its own parallel/related search. The actual implementation of this version of A* is not as a component of a web browser, but this is one potential area of application.

The proof of concept has been programmed in PERL. I attempted an object oriented design, but I think my solution is not object oriented. If I had more time, I would have more fully identified each object which collaborates to make an SA possible and to program these as parts of my SA. As it is, I have identified a few major components of the SA. I have modularized these critical parts of the SA in my solution. After reviewing literature related to Agent-Oriented Design, it became clear that a future extension of this work would profit from a further decomposition of the solution into well defined roles and responsibilities for specific objects.[57]

6.1 - Program Design

For a layout of the principal objects used to construct this program, see Figure 3 on next page. The proof of concept I am using to explore this problem has been constructed in the last 45 days. PERL was chosen as the language to test the A* implementation. This was done for 4 reasons:

1. PERL development environments include a variety of built-in functionality for processing textual information.[58]

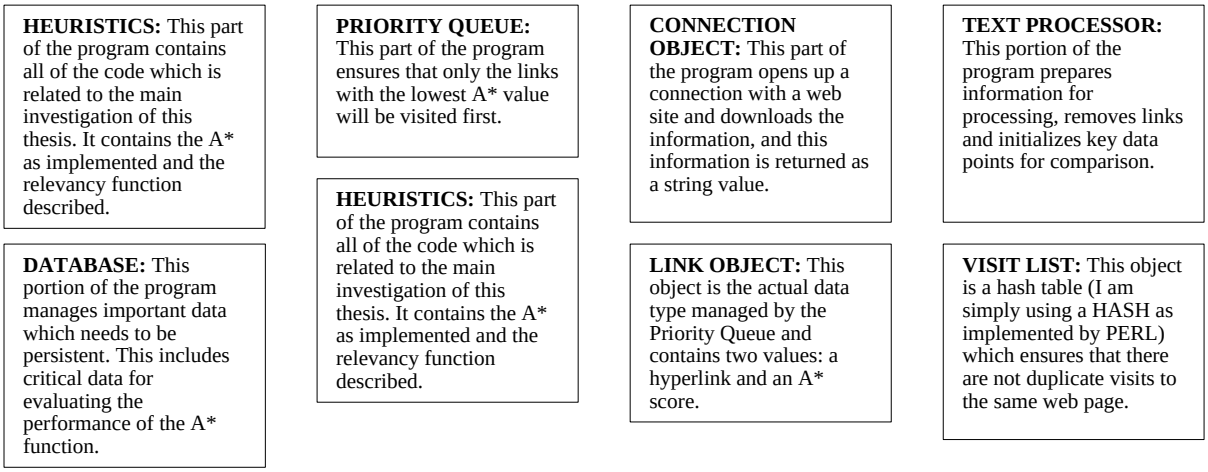
2. It is a scripting language available on many platforms and is a language accepted by academics for academic research.

3. PERL is designed for the WWW and has basic modules and extended functionality for doing many things associated with data-mining and search engines.

4. Most importantly, PERL supports a widely accepted and acknowledged standard for the implementation of **regular expressions**.[59] Since my strategy has been to deal primarily with lexical and regular structures[60], PERL makes it easy to model and filter for these kinds of structures.

[57] [Depke, Heckel and Kuster, 2001] **I think it is important to note that the underlying code could be implemented in any number of ways. At this level of abstraction we are thinking about objects in terms of what they do and what they are and not necessarily how they should be programmed – though this is very important.
[58] [Ceglowski, 2003, (internet document)]
[59] A regular expression has a more complex definition than the one I am about to give (and if you are interested you can find this definition in [Navarro and Mathieu, 2002, pg. 99]), but I think you can describe a regular expression as the following: a pattern filter which recognizes a sequence of characters but has restricted memory and cannot keep track of frequency of a character. It is pretty much the simplest way to filter information, more complex grammars make regular expressions unusable.

Figure 3: Rough Description of Objects/Modules and their Responsibilities



It is important to point out that any SA solution should only be seen in the context of a user and as part of system-user collaboration. It is believed that there may be some schism between those who would apply intelligent systems solutions to problems and those who would improve the design of user interfaces to enable people to solve problems more effectively.[61] I don't see a real contradiction between these two positions and I would contend that their debate is not taking place in the same universe of discourse. I think an SA needs to be developed with the idea of a user in mind. Even though my proof of concept does not include this as of yet, it would include this design parameter if there were the time to extend, improve and develop this work. As an experiment it is still too primitive to place in this collaborative relationship.[62]

It would be my contention that the design of an effective SA would be based upon a methodology similar to that proposed by Marvin Minsky in Society of Mind. If I had the opportunity, more time, and a large group of people interested in this problem, I would decompose the problem with a greater level of resolution – investigating all of the assumed sub-parts of making an SA and trying to model these in terms of what we understand about human cognition. The search process is very complex and there is a fair degree of synergy between the search and machine learning. And finally, as has been mentioned, there is huge potential for both

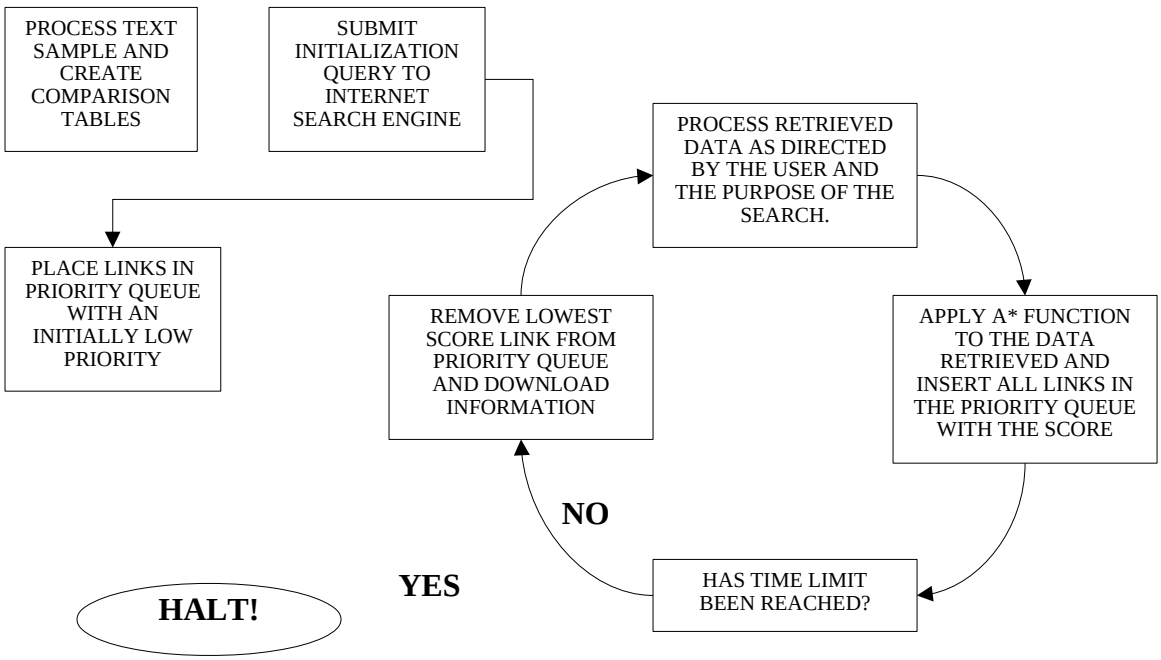
[60] I use the word regular here to refer to those abstract languages which can be generated through union, concatenation and Kleene* operations. Regular languages are the easiest to deal with, and even Context Sensitive languages like English have sub-structures which can be treated as regular even though the whole language is obviously not regular. I also think it is important to point out that Noam Chomsky, in his famous 1957 essay, Syntactic Structures, [page 22], spends the first section dispelling the notion that natural languages are regular. I reiterate this point because I don't think my position and this accepted idea are in conflict. In fact I think Chomsky would accept the notion that for pragmatic reasons context sensitive languages can be treated as context free or even regular depending upon the problem being solved.

[61] [Lieberman, 1993 (internet document)]

[62] [Maes and Schniederman, 1997, pg. 50]

the application of neural network architecture to this problem and the use of collaborative agents – agents which are homogeneous and heterogeneous. This problem can best be solved in a collaborative, cooperative and in some cases competitive space.[63] If the reader wants to get a simplified view of how the program as designed actually functions, see Figure 4 below.

Figure 4: Simplified High-Level Process Flow



6.2 - Maximum Spanning Tree Algorithm

This thesis explores the use of graphs as a representation for text documents, and attempts to use their topological properties as a basis for comparison and analysis. This work includes a minor detour into the use of maximum spanning trees for the construction of topic maps which could order the information found. Also, the use of the maximum spanning tree might be the basis for a kind of semantic marker for a document, which could be a strategy for applying new markups based upon the current and predicted protocols for the **semantic web**.[64] If there were a reasonably efficient way to automate this process, then it might not be as difficult a transition. These and other ideas surrounded the notion of using graphs for this project. The

[63] [Minsky, 1985, pg. 74]
[64] It is not necessary to cover this again; some of this is explained in the first part of this work. However, put simply the semantic web is a way of marking/representing information on the WWW to make the knowledge or meaning content of a document explicit.

literature search was not definitive for deterimining how much work has already been done using the spanning tree algorithms.

The following is a summary of potential uses of a maximum spanning tree generated from a text document:

1. Knowledge Engineering: I think there may be a way, with better pre-processing, to develop rules and predicates for a knowledge engineer automatically – based solely on the text sample. Of course this data would still need to be edited, but it might assist the knowledge engineer.

2. Automated Ontology: As already stated, I think this algorithm could be extended to assist in automatically building categorization trees. As currently designed it is not ready for this, but with sufficient text preparation it may be.

3. Associative Memory: I would like to experiment with graphs and spanning trees to explore a model for simplified associative memory. I think this could be used to build more intelligent search agents.[65]

The implementation of this is in C++ and is included in Appendix B.

7.0 - Bibliography

Books

1. Joan M. Aldous and Robin J. Wilson Graphs and Applications: An Introductory Approach, Springer-Verlag London Berlin Heidelberg, 2000

2. Avron Barr and Edward A. Feigenbaum The Handbook of Artificial Intelligence, Volume 1, Addison Wesley Publishing, 1981

[65] [Hardy, 1998, pg. 5]

3. Michael W. Berry and Murray Browne Understanding Search Engines: Mathematical Modeling and Text Retrieval, Society for Industrial and Applied Mathematics, 1999
4. Belew, Richard K. Finding Out About, Cambridge University Press, 2000
5. Berners-Lee, Tim Weaving the Web, HarperCollins San Francisco, 1999
6. Carlton McDonald and Masoud Yazdani Prolog Programming: A Tutorial Introduction, Blackwell Scientific Publications, 1990
7. Chomsky, Noam Syntactic Structures, Mouton at The Hague and Paris, 1957, 1968
8. Robert N. Moll, Michael A. Arbib and A. J. Kfoury An Introduction to Formal Language Theory, Springer-Verlag London Berlin Heidelberg, 1988
9. Hardy, Christine Networks of Meaning: A Bridget Between Mind and Matter, Praeger, Westport CN, 1998
10. Luger, George F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Essex England: Pearson Education Limited, 2002
11. Maedche, Alexander Ontology Learning For The Semantic Web, Kluwer Academic Publishers, 2002
12. Minsky, Marvin The Society of Mind, Simon and Schuster, New York, 1985
13. Gonzalo Navarro and Mathieu Raffinot Flexible Pattern Matching in Strings: Practical on-line Search Algorithms for Texts and Biological Sequences, Cambridge University Press, 2002
14. Stuart J. Russell and Peter Norvig Artificial Intelligence: A Modern Approach, Prentice Hall, Saddle River New Jersey, 1995
15. Sowa, John F. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley Publishing Company, 1984
16. Wurman, Richard S. Information Anxiety 2, Indianapolis : Que, 2001

Articles

1. Burnard, Lou “On the Intelligent Handling of Free Text Retrieval”, Oxford University Computing Service, Email Document from 29 April 1989: http://www.tei-c.org/Vault/ED/edw01.htm
2. Ceglowski, Maciej “Building a Vector Space Search Engine in Perl”, perl.com, February 19, 2003: http://www.perl.com/pub/a/2003/02/19/engine.html
3. Raymond J. Curts and Douglas E. Campbell “Avoiding Information Overload Through The Understanding Of OODA Loops: A Cognitive Hierarchy and Object Oriented Analysis and Design”, http://www.oslerbooks.com/is/pdf/ooda.pdf
4. Alan R. Dennis and Nolan J. Taylor “Information Foraging on the Web: The Effects of Internet Delays on Multi-page Information Search Behavior”, Kelley School of Business, Indiana University, August 6, 2002
5. Ralph Depke, Reiko Heckel and Jochen M. Kuster “Improving the Agent-Oriented Modeling Process by Roles”, Agents 2001, May 28 – June 1, 2001
6. Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth “The KDD Process for Extracting Useful Knowledge from Volumes of Data”, Communications of the ACM, November 1996
7. Henry Lieberman, Christopher Fry and Louis Weitzman “Why Surf Alone?: Exploring the Web with Reconnaissance Agents”, MIT Media Lab, 2000: http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Why-Surf/Why-Surf.html
8. Lieberman, Henry “Attaching Interface Agent Software to Applications”, MIT Media Lab, 1993: http://lieber.www.media.mit.edu/people/lieber/lieberary/Attaching/Attaching.html
9. Imai, Mazooki “Kaizen, the Concept”, Total Quality Handbook, Lakewood Books, 1990
10. Maes, Pattie “Intelligent Software: Programs that can act independently will ease the burdens that computers put on people”, MIT Media Lab, 1995: http://pattie.www.media.mit.edu/people/pattie/SciAm-95.html

11. Maes, Pattie “Agents that Reduce Work and Information Overload”, CACM 1994, 1994: http://pattie.www.media.mit.edu/people/pattie/CACM-94/CACM-94.p1.html
12. Pattie Maes and Ben Schniederman, “Direct Manipulation vs. Interface Agents”, Interactions, December 1997
13. Mock, Kenrick J. “Hybrid Hill-Climbing and Knowledge-Based Techniques for Intelligent News Filtering”, Intel Corporation: Intel Architecture Lab, JF2-76, Hillsboro, OR 1994: mock@cs.ucdavis.edu
14. Mathew Palakal, Snehasis Mukhopadhyay, and Javed Mostafa “An Intelligent Biological Information Management System”, Proceedings of ACM Symposium on Applied Computing, 2002
15. Rhodes and Starner “Remembrance Agent: A Continuously Running Automated Information Retrieval System”, The Proceedings of the First International Conference on the Practical Application Of Intelligent Agents and Multi Agent Technology (PAMM), 1996: 487-495
16. Sowa, John F. Automating Ontology Development, http://www.jfsowa.com/pubs/autotalk.htm, 2001
17. Sowa, John F. “Semantic Networks”, http://www.jfsowa.com/pubs/semnet.htm, 2002
18. Woods, W. A. “Finding Information on the Web: A Knowledge Representation Approach”, MIT AI Lab, 1995: http://www.ai.mit.edu/projects/iiip/conferences/www95/woods.html
19. Ora Lassila and Ralph R. Swick, editors “Resource Description Framework (RDF) Model and Syntax”, W3C Recommendation, 22 February 1999: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

Appendix A – PERL Crawler Source Code

```
#!/usr/bin/perl -w

#####################################################
### Daniel J. Sullivan, Senior Thesis Project
### I460/I461
### Applying the A* Heuristic Function to
### Document Retrieval
###
#1. This program needs the correct modules to run
#2. This program is not fully debugged
#3. There is some question as to admissibility

open(INFI, "sample.txt"); #test sample
while(<INFI>)
{
  $big_string .= $_;
}
chomp $big_string;
&Main::heuristic_search(10000,3600,0.5, $big_string, "a_star", 10000, 0.1); #10000 seems like a good

#the admissibility value and the max                    # combination
#time value should probably be very similar                                        # a good value for this
#####################################################
### Data Base Package
#####################################################
#this package is intended to manage the file i/o
#and long term storage of data generated through
#the search process.
#There is a need to ensure that the same
#links are not visited more than once.

package Data_Base;
local $file_name = '';
sub initialize
{
  $file_name = $_[0];
  mkdir "C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name, 0777;
  open(OUTFILE1, ">C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name . "_content.txt");
  print OUTFILE1 "CONTENT\n";
  close OUTFILE1;
  open(OUTFILE2, ">C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name .  "_links.txt");
  print OUTFILE2 "LINKS\n";
  close OUTFILE2;
  open(OUTFILE3, ">C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name .  "_ontology.txt");
  print OUTFILE3 "ONTOLOGY\n";
  close OUTFILE3;
```

```
  open(OUTFILE4, ">C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name .
"_search_performance.txt");
  print OUTFILE4 "SEARCH PERFORMANCE\n";
  print OUTFILE4 "CRITERIA|RELEVANCY_THREASHOLD|BPA|TIME_LIMIT|GAMMA|SEARCH_NAME|MAX_BANDWIDTH|
URL_ADDRESS|URL_DOC_SIZE|DATA_SAVED|A_STAR_EVAL|RELEVANCY|ORDERED_PAIR_REL|ALL_TERM_REL|
TOKEN_REL|CONTEN_WORD_REL|ELAPSED_TIME\n";
  close OUTFILE4;
}

sub write_to_search_performance
{
  my $temp = $_[0];
  open(OUTFILE4, ">>C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name .
"_search_performance.txt");
  print OUTFILE4 $temp . "\n";
  close OUTFILE4;
}


sub write_to_content_file #write link with content
{
  my $str1 = '';
  my $str2 = '';
  $str1 = $_[0];
  $str2 = $_[1];
  if(length($str2) < 20)
  {
    return;
  }
  open(OUTFILE1, ">>C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name . "_content.txt");
  $str2 =~ s/\s+/ /g;
  my $strlen = length($str2);
  print OUTFILE1 "\[" . $str1 . "\]" . "\n";
  if($strlen < 3000)
  {
    print OUTFILE1 substr($str2,0,$strlen) . "\n";
  }
  else
  {
    print OUTFILE1 substr($str2,0,3000) . "\n";
  }

  print OUTFILE1 "\n\n\n";
  close(OUTFILE1);
}

sub write_to_links_file
{
  my $str1 = $_[0];
  my $str2 = $_[1];
  open(OUTFILE2, ">>C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name . "_links.txt");
  print OUTFILE2 $str1 . " " . $str2 . "\n";
  close(OUTFILE2);
}
sub write_to_ontology_file
{
  my @val = @_;
  open(OUTFILE3, ">>C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\" . $file_name . "\\" . $file_name . "_ontology.txt");
  foreach $key (@val)
  {
    print OUTFILE3 $key . "\n";
  }
  close(OUTFILE3);
}
#####################################################
### Link Object for Holding link data in queue
#####################################################
package Link_Object;
sub new
{
  my ($classname, $lnk, $val) = @_;
  my $self = {};
  $self->{HYPER_LINK} = $lnk;
  $self->{VALUE} = $val;
```

```
  return bless $self, $classname;
}
package Priority_Queue;
  local @queue = ();
sub get_queue_size
{
  my $temp = @queue;
  return $temp;
}
sub insert_hyperlink
{
  my $temp1 = $_[0];
  my $temp2 = $_[1];
  my $temp3 = undef;
  my $temp4 = 0;

  $temp3 = new Link_Object($temp1,$temp2);
  &enqueue($temp3);
  $temp4 = @queue;
}
sub get_next_best_link #returns link
{
  my $temp = dequeue();
  return $temp->{HYPER_LINK};
}
sub enqueue
{
  my $temp = $_[0];
  push @queue, $temp;
}
sub sort_queue
{
  my @temp_array = ();
  @temp_array = sort { $a->{VALUE} <=> $b->{VALUE} } @queue;
  @queue = @temp_array;
}
sub dequeue
{
  my $temp = $queue[0];
  shift @queue;
  return $temp;
}
sub print_queue
{
  foreach $val (@queue)
  {
    print $val->{HYPER_LINK} . ' - ' . $val->{VALUE} . "\n"
  }
}
package Scheduler; #to include priority queue functionality
  local $start_time = 0;
  local $max_time = 0;
#time is in seconds
sub set_start_time
{
  $start_time = time;
}
sub get_elapsed_time
{
  return time - $start_time;
}
sub set_max_time
{
  $max_time = $_[0];
}
sub get_max_time
{
  return $max_time;
}

package Heuristics;

#package heuristic description
  #sub r_x_y
```

```
  #sub h_x
  #sub g_x
  #sub A_x

  #Assume network bandwidth of 50k/s
  #A perfect agent would only retrieve relevant material
  #Information retrieved might equal (bandwidth * time)

  #adjusting for a factor of less than optimal performance
  #we would multiply perfect performance by this factor GAMMA.

  #Best Path Adjusted (BPA) = Bandwidth * Time * Gamma,
  #where GAMMA = 1/x and x > 1.

  #G(n) = S, where S represents time elapsed since the search
  #has begun.

  #DV = Document Value = Relevance * Document_Size

  #Available Time(AT) = Max_Time - Time_Elapsed

  #Optimistic_Future_Retrieval = CR * x, x is required time to reach best outcome

  #(CR) Current_Relevant_Material = material on hand from search
  #which is relevant to search in bytes or kilobytes (as long as unit
  #of measure stays the same. This information can be maintained in a file.

  #So, the whole formula for H(n) is:

  #H(n) = ([Bandwidth*Max_Time*GAMMA]-Current_Relevant_Material)/(Relevance*Current_Doc_Size)

  #since every thing will be measured in a common measure of kilobytes, the
  #main unit of measure will be seconds.

  # A*(n) = H(n) + G(n) will give a total in seconds, and the priority
  # queue will extract the minimum.


  local $op_rel = 0;
  local $wc_rel = 0;
  local $tc_rel = 0;
  local $aw_rel = 0;
  local $total_bytes_saved = 0;
  local $gamma = 1; #gamma value, this value must be less than or equal to 1
  local $max_bandwidth = 0; #this value is critical to the computation
                            #the user is estimating what the perfect
                            #information download speed is for his/her
                            #computer.

#this function sums and averages multiple
#mappings.

sub get_wcr
{
  return $wc_rel;
}

sub get_opr
{
  return $op_rel;
}

sub get_tcr
{
  return $tc_rel;
}

sub get_awr
{
  return $aw_rel;
}

sub r_x_y
{
  my($z1, $z2, $z3, $z4, $x) = 0;
```

```
  #we are using weighting to adjust
  #the relative importance of each
  #mapping, this adjustment might be
  #performed by a learning algorithm.

  $z1 = &r_x_y_word_count(); #1 50%
  $z2 = &r_x_y_token_count();   #4  10%
  $z3 = &r_x_y_ordered_pairs(); #3 20%
  $z4 = &r_x_y_word_list(); #2 20%


  $x = (($z1 * 0.5) + ($z2 * 0.1) + ($z3 * 0.2) + ($z4 * 0.2))/100;

  #$x = ($z1 + $z2 + $z3 + $z4)/400;


  $wc_rel = $z1;
  $op_rel = $z3;
  $tc_rel = $z2;
  $aw_rel = $z4;

  return $x;
}

sub set_max_bandwidth
{
  $max_bandwidth = $_[0];
}

sub set_gamma
{
  $gamma = $_[0];
}

sub get_bytes_saved
{
  return $total_bytes_saved;
}

sub A_STAR_N
{
  my $f = 0;
  my $h = 0;
  my $g = 0;
  my $str1 = '';


  $str1 = $_[0];


  $h = &H_N($str1);
  $g = &G_N();


  $f = $g + $h;


  return $f;
}

sub H_N
{
  my($result, $bpa,$cr,$dv) = 0;
  my $str1 = '';


  $str1 = $_[0];


  $dv = &DV($str1);
  $bpa = &BPA();
  $cr = $total_bytes_saved;


  if($dv <= 0)
  {
```

```
    $dv = 0.00001;
  }

  $result = ($bpa - $cr)/$dv;

  return $result;
}

sub G_N
{
  my $temp = 0;

  $temp = &Scheduler::get_elapsed_time();

  print "Elapsed Time: " . $temp . "\n";

  return $temp;
}

sub BPA
{
  my($x,$y,$z) = 0;

  $x = &Scheduler::get_max_time();
  $y = $gamma;
  $z = $max_bandwidth; #max bandwidth

  return $x * $y * $z;
}

sub DV
{
  #dv = rel * size

  my $str1 = '';
  my $relevance = 0;
  my $x = 0;
  my $length = 0;

  $str1 = $_[0];

  &Text_Processor::generate_current_data($str1);

  $total_bytes_saved += length($str1);

  $length = length($str1);

  $x = &r_x_y();

  return $x * $length;
}

#returns difference between original
#and current document as a difference
#value between average relative
#frequencies for the ordered pair.

sub r_x_y_ordered_pairs
{
  my @y_keys = ();
  my %x_op_count = ();
  my %y_op_count = ();
  my $x_total = 0;
  my $y_total = 0;
  my ($i, $j, $h, $k);

  $i = 0;
```

```
  $j = 0;
  $h = 0;
  $k = 0;

  %y_op_count = &Text_Processor::get_current_ordered_pairs_count();
  %x_op_count = &Text_Processor::get_sample_ordered_pairs_count();
  $x_total = &Text_Processor::get_sample_ordered_pairs_total();
  $y_total = &Text_Processor::get_current_ordered_pairs_total();

  @y_keys = keys %y_op_count;

  foreach $key (@y_keys)
  {
    if(exists $x_op_count{$key})
    {
      $k++;

      $i = $y_op_count{$key}/$y_total;
      $j = $x_op_count{$key}/$x_total;

      $h += 1-(abs($i-$j)*100);
    }
  }

  if($k == 0)
  {
    return 5;
  }

  return ($h/$k)*100;
}

sub r_x_y_word_count
{
  my @y_keys = ();
  my %x_word_count = ();
  my %y_word_count = ();
  my $x_total = 0;
  my ($k, $n);

  $k = 0;
  $n = 0;

  %y_word_count = &Text_Processor::get_current_word_count();
  @y_keys = keys %y_word_count;
  %x_word_count = &Text_Processor::get_sample_word_count();
  $x_total = &Text_Processor::get_sample_word_total();

  if(@y_keys)
  {

    foreach $val (@y_keys)
    {
      $n++;

      if(exists $x_word_count{$val})
      {
        $k++;
      }
    }

    if($k > 0 && $n > 0)
    {
      return ($k/$n)*100;
    }
    else
    {
      return 5;
    }

  }
```

```
  else
  {
    return 5;
  }
}

sub r_x_y_token_count
{
  my %x_token_count = ();
  my %y_token_count = ();
  my $x_total = 0;
  my $y_total = 0;
  my ($i, $j, $h, $k);

  $i = 0;
  $j = 0;
  $h = 0;
  $k = 0;

  %y_token_count = &Text_Processor::get_current_token_count();
  $y_total = &Text_Processor::get_current_token_total();

  %x_token_count = &Text_Processor::get_sample_token_count();
  $x_total = &Text_Processor::get_sample_token_total();

  foreach $key (sort keys %y_token_count)
  {
    if(exists $x_token_count{$key})
    {
      $k++;

      $i = $y_token_count{$key}/$y_total;
      $j = $x_token_count{$key}/$x_total;

      $h += 1-(abs($i-$j)*20);
    }
  }

  if($h > 0 && $k > 0)
  {
    return ($h/$k)*100;
  }
  else
  {
    return 5;
  }
}

sub r_x_y_word_list
{
  my %x_count = ();
  my %y_count = ();
  my $x_total = 0;
  my $y_total = 0;
  my ($i, $j, $h, $k);

  $i = 0;
  $j = 0;
  $h = 0;
  $k = 0;

  %y_count = &Text_Processor::get_current_word_list();
  $y_total = &Text_Processor::get_current_word_list_total();

  %x_count = &Text_Processor::get_sample_word_list();
  $x_total = &Text_Processor::get_sample_word_list_total();

  foreach $key (sort keys %y_count)
  {
    if(exists $x_count{$key})
    {
      $k++;

      $i = $y_count{$key}/$y_total;
```

```
      $j = $x_count{$key}/$x_total;

      $h += 1-(abs($i-$j)*20);
    }
  }

  if($h > 0 && $k > 0)
  {
    return ($h/$k)*100;
  }
  else
  {
    return 5;
  }
}

package Text_Processor;

  use HTML::FormatText;
  use HTML::Parse;
  use HTML::LinkExtor;

  local $original_sample_document_string = ''; #original text w/o html
  local %sample_token_count = (); #set of tokens with frequency
  local $sample_token_total = 0;
  local %sample_word_count = (); #set of words with frequency
  local $sample_word_total = 0;  #relative frequency
  local %sample_ordered_pairs_count = ();
  local $sample_ordered_pairs_total = 0;
  local $translated_sample_total_words = 0;
  local %sample_unfiltered_wordlist = ();
  local $sample_unfiltered_wordlist_total = 0;

  local $original_current_document_string = ''; #current text w/o html
  local %current_token_count = (); #set of tokens with frequency
  local $current_token_total = 0;
  local %current_word_count = (); #set of words with frequency
  local $current_word_total = 0;  #relative frequency
  local %current_ordered_pairs_count = ();
  local $current_ordered_pairs_total = 0;
  local $translated_current_total_words = 0;
  local %current_unfiltered_wordlist = ();
  local $current_unfiltered_wordlist_total = 0;

#functions for accessing and processing original document characteristics

sub get_sample_ordered_pairs_total
{
  my $temp = $sample_ordered_pairs_total;

  return $temp;
}

sub get_sample_ordered_pairs_count
{
  my %temp = %sample_ordered_pairs_count;

  return %temp;
}

sub get_sample_token_count
{
  my %temp = %sample_token_count;

  return %temp;
}

sub get_sample_token_total
{
  my $temp = $sample_token_total;

  return $temp;
```

```
}

sub get_sample_total_words
{
  my $temp = $translated_sample_total_words;
  return $temp;
}

sub get_sample_word_count
{
  my %temp = %sample_word_count;

  return %temp;
}

sub get_sample_word_total
{
  my $temp = $sample_word_total;

  return $temp;
}

sub get_original_sample_document_string
{
  my $temp = $original_sample_document_string;

  return $temp;
}

sub get_sample_word_list
{
  return %sample_unfiltered_wordlist;
}

sub get_sample_word_list_total
{
  return $sample_unfiltered_wordlist_total;
}

sub generate_sample_wordlist
{
  my $str1 = $_[0];
  my @temp = ();

  $str1 =~ s/(\'|\`)//g; #removing apostrophes

  $str1 =~ s/\W+/ /g;

  @temp = split(/\s+/,$str1);

  foreach $val (@temp)
  {
    $sample_unfiltered_wordlist_total++;

    if(exists $sample_unfiltered_wordlist{$val})
    {
      $sample_unfiltered_wordlist{$val}++;
    }
    else
    {
      $sample_unfiltered_wordlist{$val} = 1;
    }
  }
}

sub generate_sample_data
{
  my $str1 = '';
  my $str2 = '';
  my $str3 = '';
  my $str4 = '';
```

```
my @temp_array = ();
my @temp_array2 = ();
my $size = 0;

$str1 = $_[0];

$original_sample_document_string = $str1;

&generate_sample_wordlist($str1);

@temp_array = &text_substitution($str1);

$translated_sample_total_words = @temp_array;

foreach $val (@temp_array)
{
  if($val =~ m/\[\d+\]/)
  {
    $sample_token_total++;

    if(exists $sample_token_count{$val})
    {
      $sample_token_count{$val}++;
    }
    else
    {
      $sample_token_count{$val} = 1;
    }
  }
  else
  {
    $sample_word_total++;

    if(exists $sample_word_count{$val})
    {
      $sample_word_count{$val}++;
    }
    else
    {
      $sample_word_count{$val} = 1;

      push @temp_array2, $val;
    }
  }
}

#generate ordered pairs

$size = @temp_array2;

my %sample_ordered_pairs_count = ();

my @temp_arr3 = ();

for($i = 0; $i < $size - 1; $i++)
{
  $j = $i;

  $str3 = $temp_array2[$j];
  $str4 = $temp_array2[$j+1];

  if($str3 ne $str4)
  {
    $str2 = $str3 . "-" . $str4;

    if((length($str3) > 3) && (length($str4) > 3))
    {
      push @temp_arr3, $str3 . "--" . $str4;
```

```
      }

      $sample_ordered_pairs_total++;

      if(exists $sample_ordered_pairs_count{$str2})
      {
        $sample_ordered_pairs_count{$str2}++;
      }
      else
      {
        $sample_ordered_pairs_count{$str2} = 1;
      }
    }
  }

  &Data_Base::write_to_ontology_file(@temp_arr3);
}

#end of functions for accessing and processing original document

#functions for accessing and processing current document

sub get_current_ordered_pairs_total
{
  my $temp = $current_ordered_pairs_total;

  return $temp;
}

sub get_current_ordered_pairs_count
{
  my %temp = %current_ordered_pairs_count;

  return %temp;
}

sub get_current_token_count
{
  my %temp = %current_token_count;

  return %temp;
}

sub get_current_token_total
{
  my $temp = $current_token_total;

  return $temp;
}

sub get_current_total_words
{
  my $temp = $translated_current_total_words;

  return $temp;
}

sub get_current_word_count
{
  my %temp = %current_word_count;

  return %temp;
}

sub get_current_word_total
{
  my $temp = $current_word_total;

  return $temp;
}

sub get_original_current_document_string
```

```
{
  my $temp = $original_current_document_string;

  return $temp;
}

sub get_current_word_list
{
  return %current_unfiltered_wordlist;
}

sub get_current_word_list_total
{
  return $current_unfiltered_wordlist_total;
}

sub generate_current_wordlist
{
  my $str1 = $_[0];
  my @temp = ();

  $str1 =~ s/(\'|\`)//g; #removing apostrophes

  $str1 =~ s/\W+/ /g;

  @temp = split(/\s+/,$str1);

  foreach $val (@temp)
  {
    $current_unfiltered_wordlist_total++;

    if(exists $current_unfiltered_wordlist{$val})
    {
      $current_unfiltered_wordlist{$val}++;
    }
    else
    {
      $current_unfiltered_wordlist{$val} = 1;
    }
  }
}

sub reset_current
{
  $original_current_document_string = ''; #current text w/o html
  %current_token_count = (); #set of tokens with frequency
  $current_token_total = 0;
  %current_word_count = (); #set of words with frequency
  $current_word_total = 0;  #relative frequency
  %current_ordered_pairs_count = ();
  $current_ordered_pairs_total = 0;
  $translated_current_total_words = 0;
  %current_unfiltered_wordlist = ();
  $current_unfiltered_wordlist_total = 0;
}

sub generate_current_data
{
  my $str1 = '';
  my $str2 = '';
  my $str3 = '';
  my $str4 = '';
  my @temp_array = ();
  my @temp_array2 = ();
  my $size = 0;

  $str1 = $_[0];

  $original_current_document_string = $str1;

  &generate_current_wordlist($str1);

  @temp_array = &text_substitution($str1);
```

```
  $translated_current_total_words = @temp_array;

  foreach $val (@temp_array)
  {
    if($val =~ m/\[\d+\]/)
    {
      $current_token_total++;

      if(exists $current_token_count{$val})
      {
        $current_token_count{$val}++;
      }
      else
      {
        $current_token_count{$val} = 1;
      }
    }
    else
    {
      $current_word_total++;

      if(exists $current_word_count{$val})
      {
        $current_word_count{$val}++;
      }
      else
      {
        $current_word_count{$val} = 1;
        push @temp_array2, $val;
      }
    }
  }

  my @temp_arr3 = ();

  $size = @temp_array2;

  my %current_ordered_pairs_count = ();

  for($i = 0; $i < $size - 1; $i++)
  {
    $j = $i;

    $str3 = $temp_array2[$j];
    $str4 = $temp_array2[$j+1];

    if($str3 ne $str4)
    {
      $str2 = $str3 . "-" . $str4;

      if((length($str3) > 3) && (length($str4) > 3))
      {
        push @temp_arr3, $str3 . "--" . $str4;
      }

      $current_ordered_pairs_total++;

      if(exists $current_ordered_pairs_count{$str2})
      {
        $current_ordered_pairs_count{$str2}++;
      }
      else
      {
        $current_ordered_pairs_count{$str2} = 1;
      }
    }
  }

  &Data_Base::write_to_ontology_file(@temp_arr3);
}

#end of functions for accessing and processing current document
```

```
sub text_substitution
{
  #SENTENCE FRAGMENT TRANSLATION: tokenizing
  #into content versus non-content language

  my $i = 0;
  my $j = 0;
  my $str1 = '';
  my @ret_array = ();

  $str1 = $_[0];

  my %parse_table = (
    'IMMEDIATELY' => '[1]', 'APPROPIATE' => '[2]', 'CONCERNING' => '[3]',
    'QUESTIONED' => '[4]', 'THEMSELVES' => '[5]', 'WEAKNESSES' => '[6]',
    'ANNOUNCED' => '[7]', 'BEGINNING' => '[8]', 'CONCERNED' => '[9]',
    'COUNTRIES' => '[10]', 'DENOUNCED' => '[11]', 'FOLLOWING' => '[12]',
    'IMMEDIATE' => '[13]', 'INDICATES' => '[14]', 'MEANWHILE' => '[15]',
    'NECESSARY' => '[16]', 'OURSELVES' => '[17]', 'PERSONNEL' => '[18]',
    'PROVIDING' => '[19]', 'QUESTIONS' => '[20]', 'SERIOUSLY' => '[21]',
    'SOMETHING' => '[22]', 'SOMETIMES' => '[23]', 'SOMEWHERE' => '[24]',
    'STATEMENT' => '[25]', 'THEREFORE' => '[26]', 'WHICHEVER' => '[27]',
    'ADDITION' => '[28]', 'BECOMING' => '[29]', 'CONCERNS' => '[30]',
    'DENOUNCE' => '[31]', 'FURTHEST' => '[32]', 'INCLUDED' => '[33]',
    'INCLUDES' => '[34]', 'INDICATE' => '[35]','INVOLVES' => '[36]',
    'LIKEWISE' => '[37]', 'MIDPOINT' => '[38]', 'ORDERING' => '[39]',
    'PREVIOUS' => '[40]', 'PROVIDED' => '[41]', 'QUESTION' => '[42]',
    'STARTING' => '[43]', 'STRONGER' => '[44]', 'STRONGLY' => '[45]',
    'UNLIKELY' => '[46]', 'WEAKENED' => '[47]', 'WEAKNESS' => '[48]',
    'WHATEVER' => '[49]', 'WHENEVER' => '[50]', 'YOURSELF' => '[51]',
    'AGAINST' => '[52]', 'BECAUSE' => '[53]', 'BETWEEN' => '[54]',
    'CALLING' => '[55]', 'CLEARLY' => '[56]', 'COUNTRY' => '[57]',
    'EARLIER' => '[58]', 'FURTHER' => '[59]', 'HOWEVER' => '[60]',
    'INCLUDE' => '[61]', 'INVOLVE' => '[62]', 'LARGELY' => '[63]',
    'LARGEST' => '[64]', 'NORTERN' => '[65]', 'NOTHING' => '[66]',
    'ORDERED' => '[67]', 'OVERALL' => '[68]', 'PATTERN' => '[69]',
    'PERCENT' => '[70]', 'PROVIDE' => '[71]', 'SEEMING' => '[72]',
    'STARTED' => '[73]', 'THROUGH' => '[1001]','WEAKEST' => '[74]',
    'WHEREAS' => '[75]', 'OWN' => '[1002]', 'WHETHER' => '[76]',
    'WISHING' => '[77]', 'START' => '[1003]', 'AMONG' => '[1004]',
    'KEEN' => '[1005]', 'EVERYTHING' => '[1006]', 'FACE' => '[1007]',
    'OTHERS' => '[1008]', 'GET' => '[1009]', 'WITHOUT' => '[78]',
    'WORKING' => '[79]', 'WORKING' => '[80]', 'YIELDED' => '[81]',
    'ALMOST' => '[82]', 'PM' => '[1010]', 'ET' => '[1011]',
    'ALWAYS' => '[83]', 'AROUND' => '[84]', 'BECOME' => '[85]',
    'BEFORE' => '[86]', 'CANNOT' => '[87]', 'DURING' => '[88]',
    'EITHER' => '[89]', 'FUTURE' => '[90]', 'ITSELF' => '[91]',
    'LATEST' => '[92]', 'LENGTH' => '[93]', 'LIKELY' => '[94]',
    'LITTLE' => '[95]', 'MEMBER' => '[96]', 'MERELY' => '[97]',
    'METERS' => '[98]', 'MIDDLE' => '[99]', 'MONTHS' => '[100]',
    'MOVING' => '[101]', 'NEARBY' => '[102]', 'NEARLY' => '[103]',
    'NOBODY' => '[104]', 'NORMAL' => '[105]', 'OLDEST' => '[106]',
    'ORDERS' => '[107]', 'PEOPLE' => '[108]', 'PUSHED' => '[109]',
    'RECENT' => '[110]', 'SAYING' => '[111]', 'SECOND' => '[112]',
    'SEEMED' => '[113]', 'SELVES' => '[114]', 'SHOULD' => '[115]',
    'SOUGHT' => '[116]', 'STATED' => '[117]', 'STRONG' => '[118]',
    'THEIRS' => '[119]', 'THOUGH' => '[120]', 'TRYING' => '[121]',
    'WEAKEN' => '[122]', 'WEAKER' => '[123]', 'WISHED' => '[124]',
    'WISHES' => '[125]', 'WITHIN' => '[126]', 'WORKED' => '[127]',
    'ABOUT' => '[128]', 'ABOVE' => '[129]', 'AFTER' => '[130]',
    'AHEAD' => '[131]', 'ALONG' => '[132]', 'BEGIN' => '[133]',
    'BEING' => '[134]', 'BEING' => '[135]', 'CLEAR' => '[136]',
    'COULD' => '[137]', 'DOING' => '[138]', 'EARLY' => '[139]',
    'ENDED' => '[140]', 'EVERY' => '[141]', 'GOING' => '[142]',
    'LARGE' => '[143]', 'LATER' => '[144]', 'LEAST' => '[145]',
    'MAYBE' => '[146]', 'MIGHT' => '[147]', 'MILES' => '[148]',
    'MONTH' => '[149]', 'MOVED' => '[150]', 'MOVES' => '[151]',
    'OFTEN' => '[152]', 'OLDER' => '[153]', 'ORDER' => '[154]',
    'OTHER' => '[155]', 'PRIOR' => '[156]', 'RIGHT' => '[157]',
    'SEEKS' => '[158]', 'SEEMS' => '[159]', 'SENSE' => '[160]',
    'SINCE' => '[161]', 'SMALL' => '[162]', 'STATE' => '[163]',
    'STILL' => '[164]', 'THEIR' => '[165]', 'THERE' => '[166]',
    'THESE' => '[167]', 'TRIED' => '[168]', 'UNDER' => '[169]',
```

```
  'UNTIL' => '[170]', 'WEEKS' => '[171]', 'WHERE' => '[172]',
  'WHICH' => '[173]', 'WHILE' => '[174]', 'WHOLE' => '[175]',
  'WHOSE' => '[176]', 'WORKS' => '[177]', 'WORLD' => '[178]',
  'WOULD' => '[179]', 'WRONG' => '[180]', 'YARDS' => '[181]',
  'YEARS' => '[182]', 'YIELD' => '[183]', 'YOURS' => '[184]',
  'ABLE' => '[185]', 'ADDS' => '[186]', 'ALSO' => '[187]',
  'ANTI' => '[188]', 'AWAY' => '[189]', 'BACK' => '[190]',
  'BEEN' => '[191]', 'BLOW' => '[192]', 'DATE' => '[193]',
  'DAYS' => '[194]', 'DOES' => '[195]', 'DONE' => '[196]',
  'FROM' => '[197]', 'GONE' => '[198]', 'HAVE' => '[199]',
  'HERE' => '[200]', 'INTO' => '[201]', 'LAST' => '[202]',
  'LATE' => '[203]', 'LEFT' => '[204]', 'LESS' => '[205]',
  'LIKE' => '[206]', 'LONG' => '[207]', 'LOSS' => '[208]',
  'LOTS' => '[209]', 'MAIN' => '[210]', 'MAKE' => '[211]',
  'MANY' => '[212]', 'MINI' => '[213]', 'MORE' => '[214]',
  'MOST' => '[215]', 'MOVE' => '[216]', 'MUCH' => '[217]',
  'MUST' => '[218]', 'NEAR' => '[219]', 'NEED' => '[220]',
  'NEXT' => '[221]', 'NONE' => '[223]', 'ONCE' => '[224]',
  'ONLY' => '[225]', 'ONTO' => '[226]', 'OURS' => '[227]',
  'OVER' => '[228]', 'PAST' => '[229]', 'SAID' => '[230]',
  'SAYS' => '[231]', 'SEEK' => '[232]', 'SEEN' => '[233]',
  'SELF' => '[234]', 'SENT' => '[235]', 'SOME' => '[236]',
  'STOP' => '[237]', 'THAN' => '[238]', 'THAT' => '[239]',
  'THEM' => '[240]', 'THEN' => '[241]', 'THEY' => '[242]',
  'THIS' => '[243]', 'TIME' => '[244]', 'TOLD' => '[245]',
  'TURN' => '[246]', 'USED' => '[247]', 'VERY' => '[248]',
  'WAYS' => '[249]', 'WEAK' => '[250]', 'WEEK' => '[251]',
  'WELL' => '[252]', 'WERE' => '[253]', 'WHAT' => '[254]',
  'WHEN' => '[255]', 'WHOM' => '[256]', 'WILL' => '[257]',
  'WISH' => '[258]', 'WITH' => '[259]', 'WONT' => '[260]',
  'WORK' => '[261]', 'YEAR' => '[262]', 'YOUR' => '[263]',
  'AGO' => '[264]', 'ALL' => '[265]', 'AND' => '[266]',
  'ANY' => '[267]', 'ARE' => '[268]', 'BUT' => '[269]',
  'CAN' => '[270]', 'DAY' => '[271]', 'DID' => '[272]',
  'DUE' => '[273]', 'END' => '[274]', 'FAR' => '[275]',
  'FOR' => '[276]', 'HAD' => '[277]', 'HAS' => '[278]',
  'HER' => '[279]', 'HIM' => '[280]', 'HIS' => '[281]',
  'HOW' => '[282]', 'ITS' => '[283]', 'LOW' => '[284]',
  'MAN' => '[285]', 'MAY' => '[286]', 'MEN' => '[287]',
  'MID' => '[288]', 'NEW' => '[289]', 'NON' => '[290]',
  'NOT' => '[291]', 'NOW' => '[292]', 'OLD' => '[293]',
  'ONE' => '[294]', 'OUR' => '[295]', 'OUT' => '[296]',
  'PRO' => '[297]', 'SAW' => '[298]', 'SAY' => '[299]',
  'SEE' => '[300]', 'SHE' => '[301]', 'THE' => '[302]',
  'TRY' => '[303]', 'USE' => '[305]', 'WAS' => '[306]',
  'WAY' => '[307]', 'WHO' => '[308]', 'WHY' => '[309]',
  'WON' => '[310]', 'YES' => '[311]', 'YET' => '[312]',
  'YOU' => '[313]', 'AM' => '[314]', 'AN' => '[315]',
  'AS' => '[316]', 'AT' => '[317]', 'BE' => '[318]',
  'BY' => '[319]', 'DO' => '[320]', 'GO' => '[321]',
  'HE' => '[322]', 'IF' => '[323]', 'IN' => '[324]',
  'IS' => '[325]', 'IT' => '[326]', 'ME' => '[327]',
  'NO' => '[328]', 'OF' => '[329]', 'ON' => '[330]',
  'OR' => '[331]', 'SO' => '[332]', 'TO' => '[333]',
  'UP' => '[334]', 'US' => '[335]', 'WE' => '[336]',
  'A' => '[357]', 'I' => '[358]', 'GRAPHICS' => '[2233]',
  'DOWNLOAD' => '[2234]', 'REUTERS' => '[2235]',
  'WEB' => '[2236]');

for($i = 0; $i <= 9; $i++)
{
  for($j = 0; $j <= 9; $j++)
  {

    $temp_string1 = $i.','.$j;
    $temp_string2 = $i . $j;

    $str1 =~ s/$temp_string1/$temp_string2/g;

  }
}

        $str1 =~ s/\s+/ /g; #removing extra whitespace
```

```
            $str1 =~ s/(\'|\`)//g; #removing apostrophes

  $str1 =~ s/\W+/ /g; #remove all characters

  $str1 =~ s/\W(united states|american|americans)\W/ USA /ig;

  $str1 =~ s/\W+U\W+S\W+/ USA /ig; #fixing U.S. problem

  $str1 =~ s/\W(\W*U\W+N\W*|united nations)\W/ UN /ig; #fixing U.N. problem

  $str1 =~ s/\d+/ [999] /g; #replace numeric quantities

  foreach $tempstr (keys %parse_table)
  {
    $str1 =~ s/\s$tempstr\s/ $parse_table{$tempstr} /ig;
  }

  $str1 = uc($str1);

  $str1 =~ s/\s+/\n/g;

  @ret_array = split("\n",$str1);

  return @ret_array

}

sub link_remover
{
  my $base_url = $_[0];
  my $source = $_[1];
  my @element = ();
  my $elt_type = '';
  my @links = ();
  my @links2 = ();

    $parser = HTML::LinkExtor->new(undef, $base_url) or die "connect error";

    $parser->parse($source) or die "connect error";

    @links = $parser->links;

    foreach $linkarray (@links)
    {
      @element = @$linkarray;
      $elt_type = shift @element;

      while(@element)
      {
        my ($attr_name, $attr_value) = splice(@element, 0,2);
        $seen{$attr_value}++;
      }
    }

    for(sort keys %seen)
    {
      $str1 .= " " . $_;
    }

    $str1 =~ s/\:\/\// /g;
    $str1 =~ s/\:/ /g;
    $str1 =~ s/\+/ /g;
    $str1 =~ s/\?/ /g;
    $str1 =~ s/\&/ /g;
    $str1 =~ s/\#/ /g;
    $str1 =~ s/\=/ /g;
    $str1 =~ s/http/ /ig;
    $str1 =~ s/newstrove\.com/ /ig;
    $str1 =~ s/\.gif/ /ig;
```

```
    $str1 =~ s/\.asp/ /ig;
    $str1 =~ s/\.pdf/ /ig;
    $str1 =~ s/\.cgi/ /ig;
    $str1 =~ s/\.php/ /ig;
    $str1 =~ s/\.jpg/ /ig;
    $str1 =~ s/\.tif/ /ig;

    $str1 =~ s/\s\/((\w|\d)+)+\/\*\s/ /ig;

    @links2 = split(/\s+/, $str1);
    @links = ();

    foreach $val (@links2)
    {
      if(length($val) > 10)
      {
        push @links, $val;
      }
    }

    %cleaning = ();

    foreach $key (@links)
    {
                            if(exists $cleaning{$key})
                            {
                              $cleaning{$key}++;
                            }
                            else
                            {
                              if($key =~ m/\w+(\w*|\W*)(\.html|\.htm|\.shtml|\.jhtml|\.stm|\.cfm|\.doc|\.txt)/i)
                              {
                                        $cleaning{$key} = 1;
                              }
                            }
                }

    @links = ();

    @links = keys %cleaning;

    return @links;
}

sub filter_links
{
                my $big_string = $_[0];
                my @links = ();
                my @temp = ();
                my %cleaning = ();

  my $current_link = $_[1];

  $big_string =~ s/newstrove\.com/ /ig;

                @temp = split(' ',$big_string);

                $big_string = join('',@temp);

                @temp = split('><',$big_string);

                $big_string = join("> <",@temp);

                @temp = split('>',$big_string);

                $big_string = join("> ",@temp);

                @temp = split('<',$big_string);

                $big_string = join(" <",@temp);

                @temp = split('//',$big_string);
```

```
		$big_string = join("// ",@temp);

		@temp = split('www',$big_string);

		$big_string = join(" www",@temp);

		@temp = split(':',$big_string);

		$big_string = join(" : ",@temp);

		@temp = split('=/',$big_string);

		$big_string = join("= /",@temp);

		@intermediate = split(' ',$big_string);

		@links = grep m/(\.html|\.htm|\.shtml|\.jhtml)/i, @intermediate;

		$big_string = join(' ',@links);

		$big_string =~ s/\.html\W*/.html /ig;
		$big_string =~ s/\.shtml\W*/.shtml /ig;
		$big_string =~ s/\.jhtml\W*/.jhtml /ig;
		$big_string =~ s/\.htm(\W+|\s+)/.htm /ig;

		@links = split(' ',$big_string);


		foreach $key (@links)
		{
			if(exists $cleaning{$key})
			{
			$cleaning{$key}++;
			}
			else
			{
			if($key =~ m/\w+(\w*|\W*)(\.html|\.htm|\.shtml|\.jhtml|\.stm|\.cfm|\.doc|\.txt)/i)
			{
				$cleaning{$key} = 1;
			}
			}
		}


		@links = keys %cleaning;

		return @links;

}

sub html_filter
{

  my $html = &parse_htmlfile("C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\dump_page.txt");
  my $formatter = HTML::FormatText->new(leftmargin => 0, rightmargin => 50);
  my $ascii = $formatter->format($html);


  $ascii =~ s/\|/ /ig;
  $ascii =~ s/\_/ /ig;
  $ascii =~ s/\-/ /ig;
  $ascii =~ s/\su\.s\.\s+/ US /ig;
  $ascii =~ s/\sp\.m\.\s+/ PM /ig;
  $ascii =~ s/\sa\.m\.\s+/ PM /ig;

  $ascii =~ s/\sjan\.\s/ january /ig;
  $ascii =~ s/\sfeb\.\s/ february /ig;
  $ascii =~ s/\smar\.\s/ march /ig;
  $ascii =~ s/\sapr\.\s/ april /ig;
  $ascii =~ s/\smay\.\s/ may /ig;
  $ascii =~ s/\sjun\.\s/ june /ig;
  $ascii =~ s/\sjul\.\s/ july /ig;
  $ascii =~ s/\saug\.\s/ august /ig;
  $ascii =~ s/\ssep\.\s/ september /ig;
  $ascii =~ s/\soct\.\s/ october /ig;
```

```
  $ascii =~ s/\snov\.\s/ november /ig;
  $ascii =~ s/\sdec\.\s/ december /ig;

  $ascii =~ s/\sh\.j\.r\.\s+/ PM /ig;
  $ascii =~ s/\.gov/ gov /ig;
  $ascii =~ s/\.mil/ mil /ig;
  $ascii =~ s/\su\.n\.\s/ UN /ig;
  $ascii =~ s/\sdr\.\s*/ doctor /ig;
  $ascii =~ s/\ssecy\.\s*/ secretary /ig;
  $ascii =~ s/\sdoc\.\s*/ document /ig;
  $ascii =~ s/\se\.g\.\s/ for example /ig;
  $ascii =~ s/(\(|\))+/ - /ig;
  $ascii =~ s/\[[^\]]*\]/ /gs;
  $ascii =~ s/<[^>]*>/ /gs;
  $ascii =~ s/\W*\*\s+/\. /ig;
  $ascii =~ s/\s+/ /ig;
  $ascii =~ s/\.\.\n/ig;

  return $ascii;
}

sub html_filter2
{
  my $ascii = $_[0];

  $ascii =~ s/<[^>]*>//gs;

  $ascii =~ s/&quot;//igs;
  $ascii =~ s/ //igs;
  $ascii =~ s/\|/ /ig;
  $ascii =~ s/\_/ /ig;
  $ascii =~ s/\-/ /ig;
  $ascii =~ s/\su\.s\.\s+/ US /ig;
  $ascii =~ s/\sp\.m\.\s+/ PM /ig;
  $ascii =~ s/\sa\.m\.\s+/ PM /ig;
  $ascii =~ s/\sjan\.\s/ january /ig;
  $ascii =~ s/\sfeb\.\s/ february /ig;
  $ascii =~ s/\smar\.\s/ march /ig;
  $ascii =~ s/\sapr\.\s/ april /ig;
  $ascii =~ s/\smay\.\s/ may /ig;
  $ascii =~ s/\sjun\.\s/ june /ig;
  $ascii =~ s/\sjul\.\s/ july /ig;
  $ascii =~ s/\saug\.\s/ august /ig;
  $ascii =~ s/\ssep\.\s/ september /ig;
  $ascii =~ s/\soct\.\s/ october /ig;
  $ascii =~ s/\snov\.\s/ november /ig;
  $ascii =~ s/\sdec\.\s/ december /ig;
  $ascii =~ s/\sh\.j\.r\.\s+/ PM /ig;
  $ascii =~ s/\.gov/ gov /ig;
  $ascii =~ s/\.mil/ mil /ig;
  $ascii =~ s/\su\.n\.\s/ UN /ig;
  $ascii =~ s/\sdr\.\s*/ doctor /ig;
  $ascii =~ s/\ssecy\.\s*/ secretary /ig;
  $ascii =~ s/\sdoc\.\s*/ document /ig;
  $ascii =~ s/\se\.g\.\s/ for example /ig;
  $ascii =~ s/(\(|\))+/ - /ig;
  $ascii =~ s/\[[^\]]*\]/ /gs;
  $ascii =~ s/<[^>]*>/ /gs;
  $ascii =~ s/\W*\*\s+/\. /ig;
  $ascii =~ s/\s+/ /ig;
  $ascii =~ s/\.+/\./ig;

  return $ascii;
}

sub html_filter3
{
  my $ascii = $_[0];
  my @arr = ();
  my @arr2 = ();

  $ascii =~ s/\n+/ /gs;
  $ascii =~ s/\s+/ /gs;
```

```
  $ascii =~ s/>\s*</>\n</gs;

  $ascii =~ s/<[^>]*>//gs;

  @arr = split(/\n/,$ascii);

  foreach $val (@arr)
  {
    if($val =~ m/\.$/)
    {
      push @arr2, $val;
    }
  }

  $ascii = join(' ',@arr2);


  $ascii =~ s/ //igs;

  return $ascii;
}

package Connection_Object;

  use LWP::UserAgent;
  use HTTP::Request;
  use HTTP::Response;
  use URI::Heuristic;
# for seeding the search use : 'search.yahoo.com/bin/search?p=north+korea'

sub connect_to_link
{
        my $raw_url = $_[0];
  my $content = '';
  my $bytes = 0;
  my $count = 0;

        my $url = URI::Heuristic::uf_urlstr($raw_url);

        $| = 1;

        printf "%s =>\n\t", $url;

        my $ua = LWP::UserAgent->new();

        $ua->agent("Schmozilla/v9.14 Platinum");

        my $req = HTTP::Request->new(GET => $url);

        $req->referer("http://cooldude.tv");

        my $response = $ua->request($req);

        if($response->is_error())
        {
        printf " %s\n", $response->status_line;
        }
        else
        {
    $content = $response->content();
    $bytes = length $content;
    $count = ($content =~ tr/\n/\n/);
    printf "%s (%d lines, %d bytes)\n", $response->title(), $count, $bytes;
        }

  return $content;
}

package Main;

sub heuristic_search
{

  my @start_queries = ();
```

```
my @temp_arr = ();
my $str1 = '';
my @visit_list = ();
my %return_word_hash = ();

my $criteria = $_[0]; #what is immediately not-admissible
my $time_limit = $_[1]; #total seconds for search
my $gamma_setting = $_[2]; #value less than or equal to 1
my $sample_text = $_[3]; #sample text used for comparison
my $file_name = $_[4]; #special prefix name for search
my $max_bandwidth = $_[5]; #setting an estimate of maximum bandwidth
my $set_relevancy = $_[6]; #relevancy threashold


my $search_data = '';

&Heuristics::set_max_bandwidth($max_bandwidth);
&Heuristics::set_gamma($gamma_setting);
&Data_Base::initialize($file_name);


push @visit_list, 'XXXX';

&Text_Processor::generate_sample_data($sample_text);

#############################################################
#### Generate query strings
#############################################################
### this is an attempt to cast a wide net for nodes
### to seed the search process.


%return_word_hash = &Text_Processor::get_sample_word_count();

foreach $key (sort {$return_word_hash{$a} <=> $return_word_hash{$b}} keys %return_word_hash)
{
  push @temp_arr, $key;
}


$m = 0;


my $s1 = pop @temp_arr;
my $s2 = pop @temp_arr;
my $s3 = pop @temp_arr;
my $s4 = pop @temp_arr;
my $s5 = $s1 . "+" . $s2 . "+" . $s3 . "+" . $s4 . "+";

while(@temp_arr && $m < 20)
{
  my $k = 0;
  $str1 = '';
  $str1 = pop @temp_arr;

  while(@temp_arr && $k < 4)
  {
    $str1 .= "+";
    $str1 .= pop @temp_arr;
    $k++;
  }

  $m++;

  my $str6 = "search.yahoo.com\/bin\/search?p=" . $s5 . $str1;

  &Priority_Queue::insert_hyperlink($str6,50000);

}

my $w = 0;

while($url_load = &Priority_Queue::get_next_best_link())
{
  if($w == 15)
  {
    last;
```

```
  }

  $str7 = &Connection_Object::connect_to_link($url_load);

  @links2 = &Text_Processor::filter_links($str7);

  foreach $val (@links2)
  {
    &Priority_Queue::insert_hyperlink($val, 10000);
  }

  $w++;
}

&Priority_Queue::sort_queue();

&Scheduler::set_max_time($time_limit);

&Scheduler::set_start_time();

my $page_val = 0;
my $str2 = '';
my $str3 = '';
my @links = ();
my $url = '';

system('cls');

$search_data =  $criteria . ", " .
                $set_relevancy . ", " .
                &Heuristics::BPA() . ", " .
                $time_limit . ", " .
                $gamma_setting . ", " .
                $file_name . ", " .
                $max_bandwidth;

my $ret_search_data = '';

my $e_time = 0;

while($e_time < &Scheduler::get_max_time())
{

  $e_time = &Scheduler::get_elapsed_time();

  $ret_search_data = '';

  open(OTEMP, ">C:\\senior_thesis\\perl_thesis_work\\crawler\\database\\dump_page.txt");

  print "\n\n";

  print "_____________________________________________________\n";

  my $q_size = &Priority_Queue::get_queue_size();

  print "Size of Queue= " . $q_size . "\n";

  $url = &Priority_Queue::get_next_best_link();

  my $flag1 = 0;

  foreach $val (@visit_list)
  {
    if($url eq $val)
    {
      $flag1 = 1;
      last;
    }
  }
```

```
if($flag1)
{
  close OTEMP;
  next;
}

push @visit_list, $url;

$str2 = &Connection_Object::connect_to_link($url);

if(length($str2) < 100)
{
  $flag1 = 1;
}

if($flag1)
{
  $ret_search_data = $search_data . '', '' . $url . ''*****BAD URL*****'';

  &Data_Base::write_to_search_performance($ret_search_data);

  close OTEMP;

  next;
}

@links = &Text_Processor::link_remover($url, $str2);

if($str2)
{
  print OTEMP $str2;
}

my $str4 = &Text_Processor::html_filter();

if(length($str4) < 1000)
{
  $str3 = uc(&Text_Processor::html_filter3($str2));
}
else
{
  $str3 = uc($str4);
}

if($str3)
{

  print '************************' . ''\n'';
  print ''       CONTENT         \n'';
  print '************************' . ''\n'';

  my $strlen = length($str3);

  if($strlen < 1000)
  {
    print substr($str3,0,$strlen) . ''\n'';
  }
  else
  {
    print substr($str3,0,1000) . ''\n'';
  }
}

print ''\n\n'';

if($url && $str3)
{
```

```
  &Data_Base::write_to_content_file($url, $str3);
}

$page_val = &Heuristics::A_STAR_N($str3);

print "A* Evaluation for this page= " . $page_val . "\n";

my $rel = &Heuristics::r_x_y();

print "Relevancy for this page= " . $rel . "\n";

print "_____\n\n";

my %ret_hash = &Text_Processor::get_current_word_count();
my @temp_arr2 = ();

foreach $key (sort {$ret_hash{$a} <=> $ret_hash{$b}} keys %ret_hash)
{
  push @temp_arr2, $key;
}

$m = 0;
my $str11 = '';

$str11 = pop @temp_arr2;

while(@temp_arr2 && $m < 20)
{

  $str11 .= "--";
  $str11 .= pop @temp_arr2;
  $m++;

}

$ret_search_data =  $search_data . ", " . $url .
                    ", " . length($str3) .
                    ", " . &Heuristics::get_bytes_saved() .
                    ", " . $page_val .
                    ", " . $rel .
                    ", " . &Heuristics::get_opr() .
                    ", " . &Heuristics::get_awr() .
                    ", " . &Heuristics::get_tcr() .
                    ", " . &Heuristics::get_wcr() .
                    ", " . $e_time;

&Data_Base::write_to_search_performance($ret_search_data);

if($str11)
{
  &Data_Base::write_to_links_file($url,$str11);
}
else
{
  my $ii = $s5;

  $ii =~ s/\+/\-\-/g;

  &Data_Base::write_to_links_file($url,$ii);
}

if($page_val < $criteria || $rel < $set_relevancy)
{
  close OTEMP;
  &Text_Processor::reset_current();
}
else
{
```

```
    foreach $val (@links)
    {
      &Priority_Queue::insert_hyperlink($val,$page_val);
    }

    &Priority_Queue::sort_queue();

    &Text_Processor::reset_current();

    close OTEMP;
   }
  }

  #clear out link queue so all links go to file

  my $uu = $s5;

  $uu =~ s/\+/\-\-/g;

  while(&Priority_Queue::get_queue_size() > 0)
  {
    my $hhh = &Priority_Queue::get_next_best_link();

    &Data_Base::write_to_links_file($hhh,$uu);
  }
}
```

Appendix B – C++ Graph Algorithm Source Code

```
#include <iostream>
#include <string.h>
#include <string>
#include <vector>
#include <queue>
#include <cstdio>
#include <cstdlib>
#include <fstream>

#define FILENAME 100

using namespace std;

class DataItem
{
        friend bool operator<(const DataItem &di1, const DataItem &di2)
        {
                return (di1.wordCount < di2.wordCount);
```

```
	}
	public:
	string word;
	float wordCount;

	DataItem(string str1) : word(str1)
	{
	}
	DataItem()
	{
	}
	DataItem(string str1, float wc) : word(str1), wordCount(wc)
	{
	}
	~DataItem()
	{
	}
};

class StringPQ
{
	private:
	priority_queue<DataItem> strings;
	int maxSize;
	int currentSize;
	public:
	StringPQ(int ms) : maxSize(ms), currentSize(0)
	{
	}
	void enqueue(string str, float wc)
	{
		if(currentSize >= maxSize)
		{
			cout << "The queue is full!!!" << endl;
			return;
		}
		DataItem da(str,wc);
		strings.push(da);
		currentSize++;
	}
	DataItem dequeue()
	{
		DataItem da = strings.top();
		strings.pop();
		return da;
	}
	string getString()
	{
		DataItem da = dequeue();
		return da.word;
	}
	void unloadAndPrintPQ()
	{
		while(!strings.empty())
		{
			DataItem da = strings.top();
			strings.pop();
			cout << da.word << " " << da.wordCount << endl;
		}
	}
	void testPQ()
	{
		ofstream out("testPQ.txt");
		while(!strings.empty())
		{
			DataItem da = strings.top();
			strings.pop();
			out << da.word << " " << da.wordCount << endl;
		}
		out.close();
	}
	void unloadPQ()
	{
		while(!strings.empty())
```

```
                {
                        DataItem da = strings.top();
                        strings.pop();
                }
        }
        bool isEmpty()
        {
                return strings.empty();
        }
        ~StringPQ()
        {
                unloadPQ();
        }
};

class StringFilter
{
        private:
        vector<DataItem*> hashArray;
        int arraySize;
        DataItem* pNonItem;
        int totInTable;
        public:
        StringFilter() : arraySize(10000)
        {
                hashArray.resize(arraySize);
                for(int j = 0; j < arraySize; j++)
                {
                        hashArray[j] = NULL;
                }
                pNonItem = new DataItem("EOF");
                totInTable = 0;
                ifstream in("tokens.txt");
                while(in)
                {
                        string temp;
                        in >> temp;
                        insert(temp);
                }
                in.close();
        }
        int getSize()
        {
                return arraySize;
        }
        void show()
        {
                cout << "Table: " << endl;
                for(int j = 0; j < arraySize; j++)
                {
                        if(hashArray[j] != NULL) cout << hashArray[j]->word << endl;
                        else
                        cout << "*** END OF LIST ***" << endl;
                }
        }
        int getTot()
        {
                return totInTable;
        }
        int hashFunc2(int key)
        {

                return 5 - key % 5;
        }
        int hashFunc1(int key)
        {
                return key % arraySize;
        }
        void insert(string kkey)
        {

                DataItem* pItem;
                pItem = new DataItem(kkey);
                int key = getKey(kkey);
```

```
		int hashVal = hashFunc1(key);
		int stepSize = hashFunc2(key);
		while(hashArray[hashVal] != NULL && hashArray[hashVal]->word != "EOF")
		{
			hashVal += stepSize;
			hashVal %= arraySize;
		}
		hashArray[hashVal] = pItem;
		totInTable++;
}
void insert(DataItem *aItem)
{
		if(isInList(aItem->word)) return;
		int key = getKey(aItem->word);
		int hashVal = hashFunc1(key);
		int stepSize = hashFunc2(key);
		while(hashArray[hashVal] != NULL && hashArray[hashVal]->word != "EOF")
		{
			hashVal += stepSize;
			hashVal %= arraySize;
		}
		hashArray[hashVal] = aItem;
		totInTable++;
}
DataItem* remove(string kkey)
{
		int key = getKey(kkey);
		int hashVal = hashFunc1(key);
		int stepSize = hashFunc2(key);
		while(hashArray[hashVal] != NULL)
		{
			if(hashArray[hashVal]->word == kkey)
			{
			DataItem* pTemp = hashArray[hashVal];
			hashArray[hashVal] = pNonItem;
			return pTemp;
			}
		hashVal += stepSize;
		hashVal %= arraySize;
		}

		return NULL;
}
DataItem* find(string kkey)
{
		int key = getKey(kkey);
		int hashVal = hashFunc1(key);
		int stepSize = hashFunc2(key);
		while(hashArray[hashVal] != NULL)
		{
			if(hashArray[hashVal]->word == kkey) return hashArray[hashVal];
			hashVal += stepSize;
			hashVal %= arraySize;
		}


		return NULL;
}
bool isInList(string value)
{
		DataItem* x;
		x = find(value);
		if(x != NULL) return true;
		else return false;
}
int getKey(string input)
{
		int hashKEY = 0;
		for(int j = 0; j < input.length(); j++)
		{
			int letter = input[j] - 96;
			hashKEY = hashKEY * 27 + letter;
		}
		int x = abs(hashKEY);
		return x;
```

```
	}
	string getAt(int x)
	{
		return hashArray[x]->word;
	}
	bool isAt(int x)
	{
		if(hashArray[x] != NULL) return true;
		return false;
	}
	void print()
	{
		for(int j = 0; j < arraySize; j++)
		{
			if(hashArray[j]) cout << hashArray[j]->word << endl;
		}
	}
}; // end of class definition for StringFilter
class EdgePairs
{
	public:
	string start;
	string end;
};
class Edge
{
	public:
	int srcVert;
	int destVert;
	int distance;
	Edge(int sv, int dv, int d)
	{
		srcVert = sv;
		destVert = dv;
		distance = d;
	}
}; // end of class edge
class PriorityQ
{
	private:
	int SIZE;
	vector<Edge*> queArray;
	int ssize;
	public:
	PriorityQ(int theSize)
	{
		SIZE = theSize;
		queArray.resize(SIZE);
		ssize = 0;
	}
	void insert(Edge *item)
	{
		int j;
		for(j = 0; j < ssize; j++)
		{
			if(item->distance >= queArray[j]->distance) break;
		}
		for(int k = ssize-1; k >= j; k--) queArray[k+1] = queArray[k];
		queArray[j] = item;
		ssize++;
	}
	Edge * removeMin()
	{
		return queArray[--ssize];
	}
	Edge * removeMax()
	{
		Edge *temp = queArray[0];
		for(int j = 0; j < ssize; j++) queArray[j] = queArray[j+1];
		ssize--;
		return temp;
	}
	void removeN(int n)
	{
```

```
                for(int j = n; j < ssize-1; j++) queArray[j] = queArray[j+1];
                ssize--;
        }
        Edge * peekMin()
        {
                return queArray[ssize-1];
        }
        int size()
        {
                return ssize;
        }
        bool isEmpty()
        {
                return (ssize == 0);
        }
        Edge * peekN(int n)
        {
                return queArray[n];
        }
        int find(int findDex)
        {
                for(int j = 0; j < ssize; j++)
                {
                        if(queArray[j]->destVert == findDex) return j;
                }
                return -1;
        }
}; // end of class priority queue
class GraphNode
{
public:

        bool isEdge;
        bool visited;
        bool isInTree;
        string nodeName;
        int nodeID;
        int wordCount;
        int outDegree;
        int edgeValue; // only applies if this is an edge
        GraphNode *nextNode;
        GraphNode *nextEdge;
        void displayNode()
        {
                cout << nodeName << "(" << nodeID << ")";
        }
        void displayEdge()
        {
                cout << "[" << edgeValue << "]" << "=" << nodeName;
        }
        GraphNode(string theName, int nid)
        {
                nodeName = theName;
                nextNode = NULL;
                nextEdge = NULL;
                edgeValue = 0;
                outDegree = 1;
                wordCount = 1;
                isEdge = false;
                visited = false;
                nodeID = nid;
        }
        GraphNode(string theName)
        {
                nodeName = theName;
                nextNode = NULL;
                nextEdge = NULL;
                edgeValue = 0;
                wordCount = 1;
                isEdge = false;
                visited = false;
                nodeID = 999999999;
        }
        ~GraphNode()
```

```
        { }
}; //
class Graph
{
friend float compareGraphs(Graph &g1, Graph &g2)
{
        // code goes here
}
private:
        GraphNode *first;
        int curID;
        int currentVert;
        int nTree;
        int nVerts;
        int INFINITY;
        const char *graphName;
        PriorityQ *thePQ;
        vector<EdgePairs> pairs;
        vector<string> thisString;
        char *inputFile;
        string stNode;
        StringPQ maxsp;
        StringPQ nodes;
        StringPQ wordList;
        queue<DataItem> topos;
        queue<string> queries;
        bool isConceptGraph;
        StringFilter sf;
        string front;


public:
        Graph(int theSize, char *name, char *infile, int QSize, bool cg) : maxsp(QSize), nodes(QSize), graphName(name),
                isConceptGraph(cg), wordList(QSize)
        {
                char cmd[FILENAME];
                strcpy(cmd, "mkdir ");
                strcat(cmd, graphName);
                system(cmd);
                stNode = "<START>";
                INFINITY = 999999999;
                first = NULL;
                curID = 0;
                currentVert = 0;
                nTree = 0;
                nVerts = 0;
                thePQ = new PriorityQ(theSize);
                thisString.resize(theSize * 2);
                char *temp;
                temp = new char[20];
                strcpy(temp,name);
                graphName = temp;
                inputFile = infile;
                if(isConceptGraph)
                {
                        buildEdges2();
                }
                else buildEdges();
                printNodes();
                for(int i = 0; i < 3; i++)
                {
                        string temp;

                        temp = wordList.getString();

                        front.append(temp);

                        front.append("+");

                }
                topologicalSort();
                maxSpanningTree();
                buildQueries();
                printGraph();
        }
```

```
~Graph()
{

}
bool isEmpty()
{
	return (first == NULL);
}
void insertNode(string key)
{
	GraphNode *newNode;
	GraphNode *previous;
	GraphNode *current;
	if(inGraph(key))
	{
		current = first;

		while(current != NULL)
		{
			if(current->nodeName == key)
			{
				current->wordCount += 1;
				return;
			}
		}
		return;
	}
	newNode = new GraphNode(key, curID);
	nVerts++;
	curID++;
	previous = NULL;
	current = first;
	while(current != NULL && key == current->nodeName)
	{
		previous = current;
		current = current->nextNode;
	}
	if(previous == NULL) first = newNode;
	else
	{
		previous->nextNode = newNode;
	}
	newNode->nextNode = current;
}
GraphNode * getNode(string key)
{

	GraphNode *current;

	current = first;

	while(current != NULL)
	{
		if(current->nodeName == key) return current;
		current = current->nextNode;
	}

	return NULL;
}
void insertStringEdge(string key, string edge)
{
	if(!getNode(key)) insertNode(key);
	if(!getNode(edge)) insertNode(edge);

	insertEdge(key,edge);
}
void insertEdge(string key, string edge)
{
	if(!inGraph(key)) return;

	GraphNode *currentNode;
```

```
		GraphNode *currentEdge;
		GraphNode *previousEdge;
		GraphNode *newEdge;
		newEdge = new GraphNode(edge);
		newEdge->isEdge = true;
		newEdge->edgeValue = 1;
		currentNode = getNode(key);
		currentNode->wordCount = currentNode->wordCount + 1;
		if(currentNode->nextEdge == NULL)
		{
			currentNode->nextEdge = newEdge;
			newEdge->nextEdge = NULL;
			return;
		}
		currentEdge = currentNode->nextEdge;
		while(currentEdge != NULL)
		{
			if(currentEdge->nodeName == edge)
			{
				currentEdge->edgeValue += 1;
				return;
			}

			if(currentEdge->nextEdge == NULL)
			{
				currentEdge->nextEdge = newEdge;
				GraphNode *temp;
				temp = getNode(key);
				temp->outDegree = temp->outDegree + 1;
				return;
			}

			currentEdge = currentEdge->nextEdge;
		}
}
GraphNode * removeNode()
{
		GraphNode *temp;
		temp = first;
		first = first->nextNode;
		return temp;
}

GraphNode * getNodeRef(int id)
{
		GraphNode *temp;
		temp = first;
		while(temp)
		{
			if(id == temp->nodeID) return temp;
			temp = temp->nextNode;
		}
		return NULL;
}

bool inGraph(string key)
{
		GraphNode *temp;
		temp = first;
		while(temp != NULL)
		{
			if(temp->nodeName == key) return true;
			temp = temp->nextNode;
		}
		return false;
}

int getDistance(int theNode, int theEdge)
{
		GraphNode *ref1, *ref2;
		GraphNode *edge;
		ref1 = getNodeRef(theNode);
		ref2 = getNodeRef(theEdge);
```

```
	edge = ref1->nextEdge;
	while(edge)
	{
		if(edge->nodeName == ref2->nodeName) return edge->edgeValue;
		edge = edge->nextEdge;
	}
	// return INFINITY;
	return 0;
}

void displayNodes()
{
	GraphNode *temp;
	temp = first;
	while(temp != NULL)
	{
		cout << temp->nodeName << "{" << temp->wordCount << "}" << endl;
		temp = temp->nextNode;
	}
}

void printNodes()
{
	char fileName1[FILENAME];
	strcpy(fileName1, graphName);
	strcat(fileName1, "/");
	strcat(fileName1, "NODELIST.txt");
	ofstream outFile1(fileName1);
	GraphNode *temp;
	temp = first;
	while(temp != NULL)
	{
		outFile1 << temp->nodeName
			<< " "
			<< temp->wordCount
			<<  endl;
		if(	temp->nodeName != "<START>" &&
			temp->nodeName != "<END>" &&
			temp->nodeName != "<COMMA>" &&
			temp->nodeName != "<QUESTION>" &&
			temp->nodeName != "<EXCLAMATION>" &&
			temp->nodeName != "<PERIOD>" &&
			temp->nodeName != "<SEMICOLON>")
		{
			wordList.enqueue(temp->nodeName, temp->wordCount);
			nodes.enqueue(temp->nodeName, temp->wordCount);
		}

		temp = temp->nextNode;
	}

	outFile1.close();
}

void printGraph()
{
	char fileName[FILENAME];
	strcpy(fileName, graphName);
	strcat(fileName, "/");
	strcat(fileName, "GRAPH.txt");
	ofstream outFile(fileName);
	outFile << "THIS IS THE GRAPH, NODES TO EDGES" << endl;
	GraphNode *currentNode;
	GraphNode *currentEdge;
	currentNode = first;
	while(currentNode != NULL)
	{
		outFile << currentNode->nodeName;
		outFile << " ";
		if(currentNode->nextEdge != NULL)
		{
			outFile << "---";
```

```
                                currentEdge = currentNode->nextEdge;
                                while(currentEdge != NULL)
                                {
                                        outFile << currentEdge->nodeName;
                                        outFile << "[" << currentEdge->edgeValue << "] ";
                                        currentEdge = currentEdge->nextEdge;
                                }
                        }
                        outFile << endl;
                        outFile << "**************************" << endl;
                        outFile << "**************************" << endl;
                        outFile << endl;
                        currentNode = currentNode->nextNode;
                }

                outFile << endl;
                outFile.close();
}

void displayGraph()
{
                cout << "This is the list of nodes in the graph with edges listed" << endl;
                GraphNode *currentNode;
                GraphNode *currentEdge;
                currentNode = first;
                while(currentNode != NULL)
                {
                        currentNode->displayNode();
                        if(currentNode->nextEdge != NULL)
                        {
                                cout << "---";
                                currentEdge = currentNode->nextEdge;
                                while(currentEdge != NULL)
                                {
                                        currentEdge->displayEdge();
                                        cout << " ";
                                        currentEdge = currentEdge->nextEdge;
                                }

                        }

                        cout << endl;
                        cout << "**************************" << endl;
                        cout << "**************************" << endl;
                        cout << endl;
                        currentNode = currentNode->nextNode;
                }
                cout << endl;
}

void topologicalSort()
{
                char fileName[FILENAME];
                strcpy(fileName, graphName);
                strcat(fileName, "/");
                strcat(fileName, "TOPOLOGICAL.txt");
                ofstream outFile(fileName);
                queue<string> theQ;
                queue<string> output;
                GraphNode *ref1, *ref2;
                theQ.push(stNode);
                while(!theQ.empty())
                {
                        ref1 = getNode(theQ.front());
                        if(ref1->visited == false)
                        {
                                output.push(theQ.front());
                                ref1->visited = true;
                                ref2 = ref1->nextEdge;
                                while(ref2)
                                {
                                        theQ.push(ref2->nodeName);
```

```
                                        ref2 = ref2->nextEdge;
                                }
                        }
                        theQ.pop();
                }
                while(!output.empty())
                {
                        GraphNode *g;
                        g = getNode(output.front());
                        outFile << output.front();
                        outFile << " ";
                        outFile << g->wordCount;
                        outFile << endl;
                        if(     output.front() != "<START>" &&
                                output.front() != "<END>" &&
                                output.front() != "<SEMICOLON>" &&
                                output.front() != "<QUESTION>" &&
                                output.front() != "<EXCLAMATION>" &&
                                output.front() != "<PERIOD>" &&
                                output.front() != "<COMMA>")
                        {
                                DataItem da(output.front());
                                topos.push(da);
                        }
                        output.pop();
                }

                setFalse();
                outFile.close();
}
void maxSpanningTree()
{
                char fileName[FILENAME];
                strcpy(fileName, graphName);
                strcat(fileName, "/");
                strcat(fileName, "MAXTREE.txt");
                ofstream outFile(fileName);
                currentVert = 0;
                while(nTree < nVerts - 1)
                {
                        GraphNode *ref;
                        ref = getNodeRef(currentVert);
                        ref->isInTree = true;
                        nTree++;
                        for(int j = 0; j < nVerts; j++)
                        {
                                if(j == currentVert) continue;
                                ref = getNodeRef(j);
                                if(ref->isInTree) continue;
                                int distance = getDistance(currentVert,j);
                                if(distance == 0) continue;
                                putInPQ(j, distance);
                        }

                        if(thePQ->size() == 0)
                        {
                                outFile << "THIS GRAPH IS NOT CONNECTED" << endl;
                                return;
                        }

                        Edge *theEdge;
                        theEdge = thePQ->removeMax();
                        int sourceVert = theEdge->srcVert;
                        currentVert = theEdge->destVert;
                        GraphNode *ref2, *ref3;
                        ref2 = getNodeRef(sourceVert);
                        ref3 = getNodeRef(currentVert);
                        int w = getDistance(sourceVert,currentVert);
                        outFile << ref2->nodeName;
                        outFile << " " << w << " ";
                        outFile << ref3->nodeName;
                        outFile << endl;
                        string tmp1 = ref2->nodeName;
                        string tmp2 = ref3->nodeName;
```

```
                        if(!(       tmp1 == "<START>" ||
                                    tmp1 == "<END>" ||
                                    tmp1 == "<QUESTION>" ||
                                    tmp1 == "<EXCLAMATION>" ||
                                    tmp1 == "<PERIOD>" ||
                                    tmp1 == "<COMMA>" ||
                                    tmp1 == "<SEMICOLON>" ||
                                    tmp2 == "<START>" ||
                                    tmp2 == "<QUESTION>" ||
                                    tmp2 == "<EXCLAMATION>" ||
                                    tmp2 == "<PERIOD>" ||
                                    tmp2 == "<COMMA>" ||
                                    tmp2 == "<SEMICOLON>" ||
                                    tmp2 == "<END>"))
                                    {
                                            string tmp3 = ref2->nodeName + "+" + ref3->nodeName;
                                            maxsp.enqueue(tmp3,w);
                                    }
                }
                setFalse();
                outFile.close();
}
void setFalse()
{
                GraphNode *temp;
                temp = first;
                while(temp != NULL)
                {
                        temp->isInTree = false;
                        temp->visited = false;
                        temp = temp->nextNode;
                }
}
void putInPQ(int newVert, int newDist)
{
                int queueIndex = thePQ->find(newVert);
                if(queueIndex != -1)
                {
                        Edge *tempEdge;
                        tempEdge = thePQ->peekN(queueIndex);
                        int oldDist = tempEdge->distance;
                        if(oldDist > newDist)
                        {
                                thePQ->removeN(queueIndex);
                                Edge *theEdge;
                                theEdge = new Edge(currentVert, newVert, newDist);
                                thePQ->insert(theEdge);
                        }
                }
                else
                {
                        Edge *theEdge;
                        theEdge = new Edge(currentVert, newVert, newDist);
                        thePQ->insert(theEdge);
                }
}

void buildEdges()
{

                string myTok1, myTok2, myTok3;
                FILE *fpr1;
                FILE *fpw1;
                int x = 0;
                char ch;
                fpr1 = fopen(inputFile, "r");
                char fileName1[FILENAME];
                strcpy(fileName1, graphName);
                strcat(fileName1, "/");
                strcat(fileName1, "INTERMEDIATE.txt");
                fpw1 = fopen(fileName1, "w");
                fputs("<START>",fpw1);
                putc('\n',fpw1);
```

```
while(ch != EOF)
{
		ch = getc(fpr1);
		if( ch == -1) break;
		if( ispunct(ch) )
		{
			switch(ch)
			{
				case '.':
				{
					putc('\n',fpw1);
					fputs("<PERIOD>",fpw1);
					putc('\n',fpw1);
					fputs("<START>",fpw1);
					putc('\n',fpw1);
					break;
				}
				case ',':
				{
					putc('\n',fpw1);
					fputs("<COMMA>",fpw1);
					putc('\n',fpw1);
					break;
				}
				case ';':
				{
					putc('\n',fpw1);
					fputs("<SEMICOLON>",fpw1);
					putc('\n',fpw1);
					break;
				}
				case '!':
				{
					putc('\n',fpw1);
					fputs("<EXCLAMATION>",fpw1);
					putc('\n',fpw1);
					fputs("<START>",fpw1);
					putc('\n',fpw1);
					break;
				}
				case '?':
				{
					putc('\n',fpw1);
					fputs("<QUESTION>",fpw1);
					putc('\n',fpw1);
					fputs("<START>",fpw1);
					putc('\n',fpw1);
					break;
				}
			}
		}
		if( isalpha(ch) )putc(toupper(ch), fpw1);
		if( ch == ' ') putc('\n', fpw1);

}
fclose(fpr1);
fclose(fpw1);
char fileName2[FILENAME];
strcpy(fileName2, graphName);
strcat(fileName2, "/");
strcat(fileName2, "EDGES.txt");
ifstream in(fileName1);
ofstream outFile(fileName2);
while(in)
{
		in >> myTok1;
		if(myTok1 != " " || myTok1 != "")
		{
			thisString[x] = myTok1;
			x++;
		}
}
in.close();
pairs.resize(x * 2);
```

```
	for(int i = 0; i < x; i++)
	{
		pairs[i].start = "XXXX9999";
		pairs[i].end = "XXXX9999";
	}

	for(int i = 0; i < x; i++)
	{
		pairs[i].start = thisString[i];
		if(i < x) pairs[i].end = thisString[i+1];
		if(i > 0)
		{
			pairs[i+1].start = thisString[i];
			pairs[i+1].end = thisString[i-1];
		}
	}

	for(int i = 0; i < x; i++)
	{
		if	(
			pairs[i].start != "<PERIOD>" &&
			pairs[i].start != "<EXCLAMATION>" &&
			pairs[i].start != "<QUESTION>" &&
			pairs[i].end != "XXXX9999" &&
			pairs[i].end != "" &&
			pairs[i].end != " " &&
			!(pairs[i].start == "<START>" && pairs[i].end == "<START>")
			)
			{
				insertStringEdge(pairs[i].start,pairs[i].end);
				outFile << pairs[i].start;
				outFile << " ";
				outFile << pairs[i].end;
				outFile << endl;
			}
	}
	outFile.close();
}

void buildEdges2()
{
	string myTok1, myTok2, myTok3;
	FILE *fpr1;
	FILE *fpw1;
	int x = 0;
	char ch;
	fpr1 = fopen(inputFile, "r");
	char fileName1[FILENAME];
	strcpy(fileName1, graphName);
	strcat(fileName1, "/");
	strcat(fileName1, "INTERMEDIATE.txt");
	fpw1 = fopen(fileName1, "w");
	fputs("<START>",fpw1);
	putc('\n',fpw1);
	while(ch != EOF)
	{
		ch = getc(fpr1);
		if( ch == -1) break;
		if( ispunct(ch) )
		{
			switch(ch)
			{
				case '.':
				{
					putc('\n',fpw1);
					fputs("<END>",fpw1);
					putc('\n',fpw1);
					fputs("<START>",fpw1);
					putc('\n',fpw1);
					break;
				}
				case '!':
				{
```

```
                                        putc('\n',fpw1);
                                        fputs("<END>",fpw1);
                                        putc('\n',fpw1);
                                        fputs("<START>",fpw1);
                                        putc('\n',fpw1);
                                        break;
                                }
                                case '?':
                                {
                                        putc('\n',fpw1);
                                        fputs("<END>",fpw1);
                                        putc('\n',fpw1);
                                        fputs("<START>",fpw1);
                                        putc('\n',fpw1);
                                        break;
                                }
                        }
                }

                if( isalpha(ch) )putc(toupper(ch), fpw1);
                if( ch == ' ') putc('\n', fpw1);
}
fclose(fpr1);
fclose(fpw1);
// ******************************
// Filter out grammar terms
// ******************************
ifstream in1(fileName1);
ofstream out1("temp.txt");
string myGTOK;
while(in1)
{
        in1 >> myGTOK;
        if(!sf.isInList(myGTOK)) out1 << myGTOK << endl;

}
out1.close();
in1.close();
char fileName2[FILENAME];
strcpy(fileName2, graphName);
strcat(fileName2, "/");
strcat(fileName2, "EDGES.txt");
ifstream in("temp.txt");
ofstream outFile(fileName2);
while(in)
{
        in >> myTok1;
        if(myTok1 != " " || myTok1 != "")
        {
                thisString[x] = myTok1;
                x++;
        }
}
in.close();
system("rm temp.txt");
pairs.resize(x * 2);
for(int i = 0; i < x; i++)
{
        pairs[i].start = "XXXX9999";
        pairs[i].end = "XXXX9999";
}
for(int i = 0; i < x; i++)
{
        pairs[i].start = thisString[i];
        if(i < x) pairs[i].end = thisString[i+1];

        if(i > 0)
        {
                pairs[i+1].start = thisString[i];
                pairs[i+1].end = thisString[i-1];
        }


}
for(int i = 0; i < x; i++)
```

```
		{
			if	(
				pairs[i].start != "<END>" &&
				pairs[i].end != "XXXX9999" &&
				pairs[i].end != "" &&
				pairs[i].end != " " &&
				!(pairs[i].start == "<START>" && pairs[i].end == "<START>")
				)
				{
					insertStringEdge(pairs[i].start,pairs[i].end);
					outFile << pairs[i].start;
					outFile << " ";
					outFile << pairs[i].end;
					outFile << endl;
				}
		}
		outFile.close();
	}
	void buildQueries()
	{
		char fileName[FILENAME];
		strcpy(fileName, graphName);
		strcat(fileName, "/");
		strcat(fileName, "QUERIES.txt");
		ofstream outFile(fileName);
		while(!(maxsp.isEmpty() || nodes.isEmpty() || topos.empty()))
		{
			string str1 = maxsp.getString();
			string str2 = nodes.getString();
			DataItem da = topos.front();
			topos.pop();

			string str3 = da.word;

			string str4 = front + str1 + "+" + str2 + "+" + str3;

			queries.push(str4);
		}
		while(!queries.empty())
		{
			string temp = queries.front();

			outFile << temp << endl;

			queries.pop();
		}
		outFile.close();
	}

}; // end of class graph definition
int main()
{

	Graph *A;
	Graph *B;
	Graph *C;
	A = new Graph(20000, "A", "iraq_1.txt", 10000, false);
	B = new Graph(20000, "B", "iraq_2.txt", 10000, false);
	C = new Graph(20000, "C", "korea_1.txt", 10000, false);
	//delete A;
	//delete B;
	//delete C;
	return 0;
}
```